REALTIME DESIGN AND ANALYSIS OF 3D STRUCTURES USING FINITE ELEMENT ANALYSIS WITHIN VIRTUAL REALITY ENVIRONMENTS

A Thesis

by

MICHAYAL THOMAS MATHEW

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Darren Hartl
Committee Members,	John D. Whitcomb
	Ann McNamara
Head of Department,	Rodney Bowersox

August 2020

Major Subject: Aerospace Engineering

ABSTRACT

Structural analysis is a 3-dimensional concept that has traditionally been taught with 2-dimensional mediums, such as whiteboards and computer screens. This often leads to a cognitive disconnect as students are forced to rely on their individual imaginations to form complete visualizations of the topic. Understanding how structures are affected by different loading conditions is not a trivial skill for students learning the concepts. While laboratory classes can provide some real world perspective into how structures deform, they require expensive testing equipment setups and training which take away from their experiential learning capabilities.

Virtual reality is an emerging technology that can be leveraged to conduct structural analysis while intuitively teaching how it is done. Using the fundamental equations and geometric principles behind finite element analysis, a virtual environment can be created where students can experiment with creating and editing their own real-time deformable structures. To provide a true classroom experience, this simulation could run as both a local and multi-user shared experience. This will require leveraging two different virtual reality software development platforms, such as Unity and A-Frame, to create the simulation and output it to a user wearing a virtual reality headset.

Tools will need to be scripted for these experiences to allow a user to define a structure, either in pre-processing or in real time, within the experience. The user will then have the capability to modify the structure and its material properties. This will require seamless integration between the user's interactions and the finite element analysis solver to update the results with minimal latency. The integrity of the finite element analysis results from within the simulation will be numerically validated against a trusted commercial software to confirm accuracy. After analysis, the simulation will need to visualize the long slender members of the deformed structure. The current implementation will accurately render truss elements, but will need further improvements to visualize frame elements correctly. Multiple use case scenarios will be defined as an extension of this work to directly benefit students in the classroom as well as professionals who are conducting structural analysis.

DEDICATION

To my mother, Susie and my father, Philip. None of this would have been possible without the sacrifices that you have made for me and the encouragement that you have provided me throughout the years. To my brothers, Solomon and Samson, thank you for the continued support and guidance, I am who I am today because of your influences. Thank you.

“We’re not on this stage just because of talent or ability. We’re up here because of 4 A.M. We’re up here because of two-a-days or five-a-days. We’re up here because we had a dream and let nothing stand in our way. If anything tried to bring us down, we used it to make us stronger."

- Kobe Bryant

ACKNOWLEDGEMENTS

I'm truly fortunate to be where I am today and the number of people that I have to thank for aiding me in my journey are countless. I have to give glory to God for blessing me and guiding me on how to live a life of virtue. I also have to thank my parents for the countless sacrifices they have made for my development and growth. It's not easy to pack up everything and move to a new country with no idea what's waiting on the other side. Thank you for doing that so my brothers and I could get a university education. Thank you for always pushing me to become the best that I can be academically. Thank you for driving me 30+ minutes every day in middle school to attend the WAVE magnet program. Thank you for allowing me to move 6 hours away during my last two years of high school to attend the TAMS program. Thank you for financially supporting me through my time at Texas A&M. Thank you both for everything. On a similar note, I'd like to thank my brothers. It's not easy following in the footsteps of giants, but thank you for paving the path for me to succeed. Thank you for developing me and investing your time, effort and money into me. Thank you for calling me out on my mistakes, showing me how to do things better, and for constantly pushing me. I am truly grateful.

This work would not have been possible without my mentor, Dr. Darren Hartl. I was a young, bright-eyed freshman entrepreneur when he took me under his wing. I had just pitched VIZard, my idea for a virtual reality educational software when Dr. Hartl put out a very fortunate email about hiring undergraduate researchers for the summer. Our ideas for how VR technology could change the engineering education realm overlapped and he brought me on board 4 years ago to help turn that into a reality. What a wonderful ride it has been. He truly has been the greatest mentor, always pushing me to do more and bringing his ingenuity to all aspects of my work. I would not be as confident in my engineering capabilities, or even getting a masters degree if it wasn't for his belief in me and my abilities. You pushed me further than I ever thought I could go, thank you from the bottom of my heart, you changed my life. I hope to one day to become as good of a mentor to others as you were to me.

I would also like to thank Dr. Whitcomb and Dr. McNamara for being on my committee and helping me develop my work. Dr. McNamara taught me about Virtual Reality techniques that were very useful when designing my own experiences for this work. Dr. Whitcomb and his Finite Element Analysis textbook were invaluable to my understanding of this topic.

My research has greatly benefited from my community of fellow scholars. The MAESTRO lab has been an invaluable resource that has helped me develop over the years. Thank you for the kind encouragements and positivity that you have shared with me. I would like to notably thank Justin Cabazuela and David Nash for working alongside me in the VR lab. David laid the groundwork for this thesis through his many contributions to my knowledge of Unity development and programming knowledge. Thank you for being an outstanding friend and researcher, I look forward to creating that start-up with you in the future. I would also like to thank my college roommates George and Sean for being my support system and helping me through the years. Ian and Isaac Hartl designed the structure that was used as the test case for a lot of this work and I am truly grateful for their help. I would like to specifically thank Brent Bielefeldt, Jacob Mingear, Kasey Clark, Pedro Leal, and William Scholten for proofreading and providing feedback on the contents of this work.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Darren Hartl and John Whitcomb of the Department of Aerospace Engineering, and Professor Ann McNamara of the Department of Visualization.

The 3D FEA solver was initially implemented by Dr. Brent Bielefeldt and Holly Patterson for the SPIDRS program which helped motivate this work.

Initial virtual reality implementations were assisted by David Nash who enabled a lot of productive troubleshooting within the Unity environment. He was also helpful in designing user interaction capabilities as well as conducting research into networked A-Frame environments.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a grant from Texas A&M Triads for Transformation (T3) as well as Award #1829799 from the National Science Foundation (NSF) Office of Advanced Cyberinfrastructure (OAC).

NOMENCLATURE

2D	2-Dimensional
3D	3-Dimensional
VR	Virtual Reality
SDK	Software Development Kit
HMD	Head Mounted Display
FEA	Finite Element Analysis
FEM	Finite Element Method
PDE	Partial Differential Equation
σ	Stress
ε	Strain
DOF	Degrees of Freedom
$\boldsymbol{N(x)}$	Matrix of shape functions
$\boldsymbol{p^T}$	Vector of polynomial basis functions
l_e	Length of element
u_i	Displacement DOF at node i in x direction
v_i	Displacement DOF at node i in y direction
w_i	Displacement DOF at node i in z direction
θ_{xi}	Rotational DOF at node i in x direction
θ_{yi}	Rotational DOF at node i in y direction
θ_{zi}	Rotational DOF at node i in z direction
$\boldsymbol{B}$	Strain Matrix
$\boldsymbol{k_e}$	Element stiffness matrix (local coordinate system)

$\boldsymbol{f_e}$	Element force vector (local coordinate system)
$\boldsymbol{d_e}$	Element displacement vector (local coordinate system)
A_e	Cross-sectional area of element
E	Young's modulus of element material
G	Shear modulus of element material
ν	Poisson's ratio of element material
J	Cross-sectional polar moment of Inertia
ξ	Intermediary variable representing distance from the center of the element
I_z	Moment of Inertia of the cross section of the beam with respect to the z-axis
I_y	Moment of Inertia of the cross section of the beam with respect to the y-axis
$\boldsymbol{T}$	Transformation matrix relating local and global coordinate systems
$\boldsymbol{K_e}$	Element stiffness matrix (global coordinate system)
$\boldsymbol{F_e}$	Element force vector (global coordinate system)
$\boldsymbol{D_e}$	Element displacement vector (global coordinate system)
$\boldsymbol{K_g}$	Structure stiffness matrix (global coordinate system)
$\boldsymbol{F_g}$	Structure force vector (global coordinate system)
$\boldsymbol{D_g}$	Structure displacement vector (global coordinate system)

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Motivation

Structural design techniques are something that many children learn at a young age. This can be done through simple toys (Lego) or through actual structural components (bricks). Through the process of trial-and-error, the children gain an intuitive grasp on how to build structures that are strong or meet the criteria that they want. They unnervingly tackle difficult physics concepts such as gravity and momentum by adapting the toy's structural design to not fail in that same way in the next design iteration. They do not know it then, but they are conducting structural analysis and learning the basics of it. As the children get older, their structural analysis techniques are emboldened by new competitions that seek to challenge their skills. These competitions are phrased simplistically, such as; design a bridge using spaghetti and marshmallows, or design a capsule that won't break the egg on an impact drop. However, the complex goal is to strengthen the students' structural analysis intuition. If these students decide to go to university and pursue engineering, they are given a more mathematical understanding of structural analysis that builds on and confirms their childhood intuitions.

At a university, structural analysis is traditionally taught in phases. First, the students are taught mechanics of materials, which introduces them to the theory of how solid objects react when they are subjected to stresses and strains. Then they are taught about simple structural mechanics and how the deformations and internal forces of simple structures are calculated. This methodology is then reinforced as they are taught about Finite Element Analysis (FEA), an analytical approach that subdivides a complex structure into smaller elements to approximate its real world behavior. Conducting FEA is usually left up to commercial softwares such as ABAQUS and ANSYS, and is only taught in the classroom to a certain degree. This is because of how complex visualizing these models would be on a whiteboard. This burden can be eased through the use of experiential laboratory classes, where the strength of structures can be tested using machines. However, this

proves to not be very beneficial as the necessary machines are expensive and also require intensive training in order to be used properly. This leaves students in the role of a spectator, as they watch someone else perform the testing, without ever getting to be hands-on with it themselves. While there are plenty of tutorials to follow for students with a license to the commercial FEA programs, those design experiences have a large learning curve and are still visualized on 2D screens which are not very intuitive. The primary weakness with 2D screens in the context of 3D representations is that it brings a lack of perspective when visualizing models in pseudo-3D environments [1]. When viewed from some perspectives, the image on the left in Figure 1.1 shows what seems to be an "impossible" shape: a never-ending staircase. Only the appropriate manipulation of views would allow a user to see this object as anything more than a poorly designed and useless staircase. However, the true nature of this shape can be intuited given the appropriate viewpoint, as seen by the image on the right.

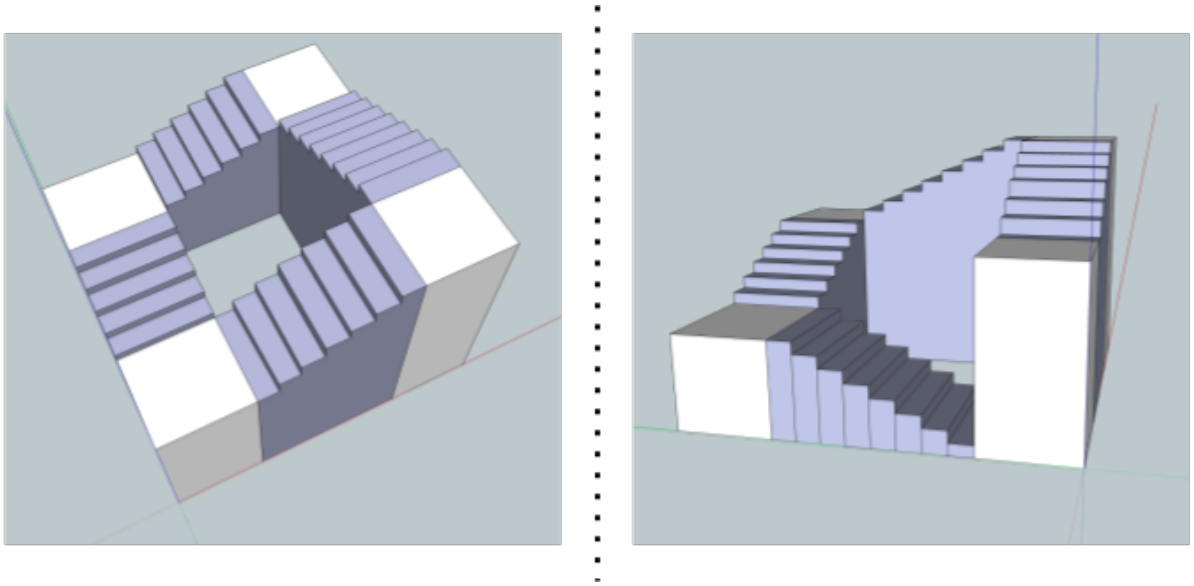


Figure 1.1: The lack of perspective with 2D screens can lead to the emergence of visual illusions

While Figure 1.1 above is a simple example of the shortcomings of 2D visualizations of 3D models, a robust and manipulation-friendly user environment can help alleviate this perspective

issue. Currently, 3D models are visualized using windows, icons, menus, pointer (WIMP)-style-based environments. These environments deprive the user of their senses of physical characteristics such as scale, orientation, etc [2]. As a result, the user will have to rely on their imagination to visualize what the model would actually look like in the natural world at true scale. Virtual Reality (VR) and Augmented Reality (AR) technologies bridge the gap by enabling depth perception and freedom of movement (e.g. perspective changes) making the spatial understanding of objects more accurate, natural, and intuitive [3]. VR provides a cost effective way to visualize structures of varying scales and to analyze how they deform under various loading conditions. While leveraging advances in Virtual Reality (VR) technology to enhance structural engineering visualization is not a novel topic [4], an immersive real time experience that enables users to design, analyze and interact with 3D structures [5] will be the topic of the research for this thesis. Initial implementation with visualizing FEA results in a VR environment revolved around designing and processing a structure in ABAQUS and exporting those results for a VR environment as seen in Figure 1.2 and discussed in Chapter 2.

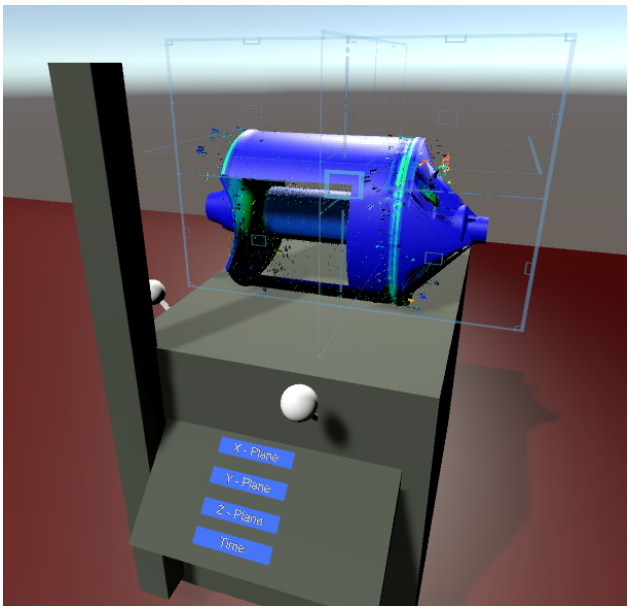


Figure 1.2: Post-processing FEA results in a Unity VR environment

Exporting results out of commercial softwares is in-line with the literature review of previ-

ous research attempts at creating this type of a design tool [6]. Most of the previous scholarly work revolved around post-processing FEA results from commercial codes and then modifying those results using VR. These modified structures would then be sent back to the FEA software to re-evaluate the new design [7]. While it was not processed in real time, these improved analysis methods were benchmarked at improving productivity 10-30 times over traditional methods [8]. The goal of this thesis is the development of a software tool to conduct real-time design and analysis of structures within Virtual Reality environments. An additional goal is the development of particular use cases that utilize this tool in educational and collaborative settings to provide users with an immersive structural analysis experience. While two of the three cases have not been tested, their preliminary design should enable expedited future developments. The elements of the deformed structure visualized through this simulation are rendered accurately for truss elements but will need to be enhanced in future work to capture the full shape deformation of frame elements.

1.2 Literature Review

1.2.1 Virtual Reality Technology

Virtual Reality as a concept has been around in science fiction novels, and research laboratories for decades. The first VR headset, the "Sword of Damocles" was made by Ivan Sutherland and was touted as the ultimate display [9]. His futuristic portrayal of computer displays that would realistically mimic the natural world served as the foundation which modern day VR headsets were built on. In essence, virtual reality is a simulation made by computer graphics to represent an alternate environment [10]. Virtual reality visualizations have been available since the 1980's with the hopes of providing a more intuitive way to see and visualize data [11, 12]. In this time, structural engineering research labs have attempted to utilize it to enhance how they conduct structural modeling [13] and visualization [14]. A couple successful VR applications for the medical field were introduced in the early 21st century [15], some even attempted to utilize finite element methods to conduct surgery simulations [16]. While the methodologies implemented were sound,

the lack of necessary technical capabilities caused these applications to sputter out. Compared to the year 2020, the technology available was drastically inferior. The computers of the time lacked processing power, graphical capabilities, and memory storage. This directly affects the fidelity of the simulations created. Computers made since 2015 are more than 1000% more capable than those of the late 1990's and early 2000's [17], and have greatly improved the capabilities of Virtual Reality simulations.

However, commercial VR devices has only recently become feasible to the masses. Since the roll out of the first accessible commercial VR headsets in 2016, there has been stiff competition to be the first to market with good immersive VR hardware and software. The VIVE headset, created by industry leader HTC, has been particularly successful due to posessing a variety of features. The VIVE has a headset and controllers capable of 6 degree of freedom (DOF) movement [18]. These peripherals are tracked by 2 base stations that flood the room with infrared light as seen in Figure 1.3. This enables the user to freely navigate around the confines of a virtual environment that is mapped to dimensions of the physical room. The user can then utilize the controllers to interact with objects in this environment and all input can be read and understood by the simulation. Previous research capabilities in virtual reality were greatly hindered by the lack of 6DOF tracking technology [19].

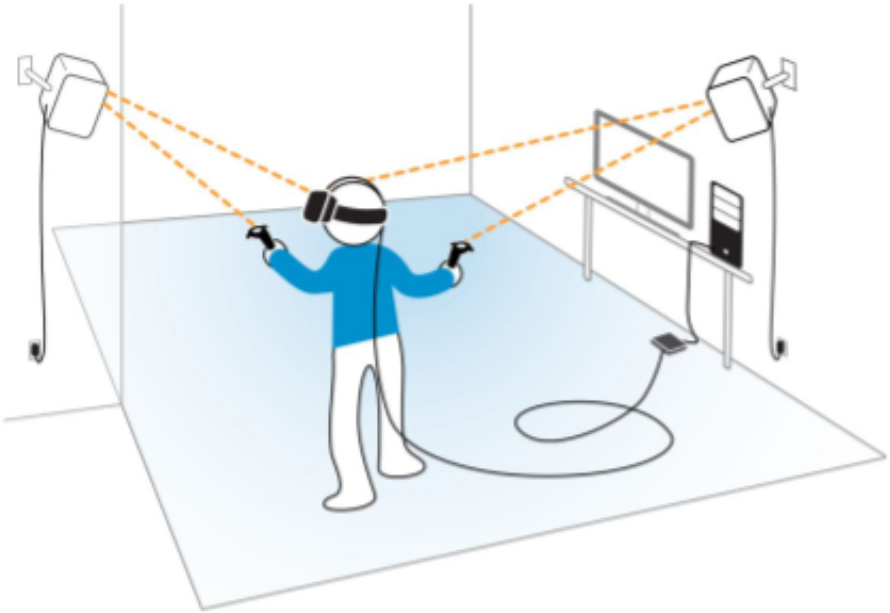


Figure 1.3: Virtual environment made possible by the 2016 HTC Vive headset [18]

1.2.2 Virtual Reality Simulation Development

There are many 3-D engines utilized in the development of VR simulations, with each engine requiring the use of their own language and mechanics. Unity is a popular engine utilized by game and simulation developers alike because of their intuitive interface and utilization of the popular $C\#$ language [20]. Within Unity, the Software Development Kit (SDK) provided by HTC for their VIVE headset can be implemented to create a tracked 3-D environment and to read the various user inputs from the controllers. Other third-party SDKs can be implemented as well, to configure items like laser pointers as well as physics. This eases the burden on the developer as they can focus on the software capabilities instead of writing hardware level code. Third-party SDKs are what make the Unity Engine an easier choice for novice and experienced developers alike [21], as more complex tools can be developed by utilizing them.

Another engine that is quickly gaining popularity but is currently under-utilized is A-Frame. A-Frame, powered by Mozilla, uses the $JavaScript$ language and a new standard called WebVR to bring all the beauty of VR simulations to your mobile or computer browser [22]. This enables

collaborative VR, allowing many people to interact with the same simulation simultaneously, while using different types of devices. The device agnostic approach to VR is seen as a breath of fresh air by developers as they can focus on developing the fidelity of their simulations, without being caught up by all the device-specific interaction jargon created by hardware makers [23]. However, a few large downsides appear from this approach. One downside arises from the fact that all programming is done generically. This means that no interactions are optimized for any devices, which leads to poor performance across the board when trying to implement complex capabilities. Another downside is that all the computing is done on board the device. This means that when the user uses a less capable device, such as a phone or tablet, it can prove to be graphically and/or computationally expensive to render, effective trading off capability for convenience.

Virtual reality simulation design concepts have been considered and implemented throughout this work. Concepts such as navigation in the virtual environment [24], selection and manipulation of objects [25, 26], user interfaces [27, 28], virtual environment design [29, 30], and collaborative experiences [31]. A common theme of keeping latency below 20 milliseconds to mitigate virtual reality sickness was found [32]. Virtual reality sickness is a problem that varies from user to user but needs to be carefully considered when designing virtual experiences. Common symptoms of this sickness are discomfort, headaches, disorientation and nausea. While there are many causes that contribute to these symptoms, one of the main contributors is the delay between when an user does an action, and when the action is actually registered, otherwise known as latency [33]. A slight increase in latency (>10ms) will cause the loss of presence (feeling of being in a virtual environment), and a larger latency (>20ms) will cause disorientation and potentially worse symptoms. Latency is usually caused by an overload of the computer's processing capabilities. A couple of solutions are to either get a more powerful computer, or to reduce the visualization complexity and keep the processing demand below a certain level at all times.

1.2.3 Structural Analysis In Virtual Reality

Real-time structural analysis within virtual environments has been heavily investigated in the past. Scherer and Wabner implemented a method to improve human perception and comprehension

of FEA results through the use of stereoscopic visualization methods [34]. The FEMvrml system used structures exported into the VRML (Virtual Reality Markup Language) format and visualized the structural analysis results [35]. They developed the ability to control and explore the results by preparing the animations required beforehand. Connell and Tullberg presented a framework for visualizing real-time responses to FEA simulations by using an approximation module that efficiently generates and shows simplified results before accurate results can be displayed [36]. Ryken and Vance developed a VR application that aids in the design of a tractor arm in a CAVE VR environment [37]. With that application, designers could modify the shape of a component and view the updated stresses interactively, this was approximated through the use of a linear interpolation of a sensitivity analysis carried out beforehand. Video games are also an avenue where the finite element method has found success. FEM has successfully been implemented in video games by pre-computing deformations [38] and visualizing them as animations or free-form deformation renderings [39].

The archival literature does not provide many examples of efforts or successes in directly embedding a finite element solver into an AR/VR application for real time analysis [40]. This is due to the computational limitations of solving for FEA results in real-time. If the structure is more complex, more time will be needed by the solver, which will then increase the latency [41]. This is undesirable because increased latency makes the users experience dizzying effects within the simulation [11]. A few alternate methods have been researched to the computational limitation of solving in real time. One such method was through the use of a neural network. After training the neural network with user data, the common configurations could be predicted. Thus the solutions to the FEA problem can be pre-computed and presented in real time as necessary [42]. Other methods include visualizing deformation using approximation methods while the FEA is being run [43, 44] and then updating the solution once completed, as well as off-loading the FEA to an external application or processing system [45, 46], and only showing the results when the data is ready [47]. Simple structures have been proven to be solved in real-time by [48] by integrating a solver tool within their virtual environment. However, their lessons learned noted the increase in latency

as the structural mesh exceeds ten elements. Current state of the art capabilities for structural analysis involve an application with a framework that implements a real time creation capability. However, there is still a wait time as the creation is then offloaded to a server for computational results, and then re-imported back to the simulation once completed [49]. A system called VirDe was developed by Ingrassio and Cappello to streamline the design process and conduct both CAD modeling and FEA simulation within the VR environment even further [50]. The VirDe system served as a conduit for viewing OpenInventor models in VR, applying boundary conditions and forces which would then conduct the analysis by sending the defined structure to ANSYS. A benefit of this off-loading method is that solving systems with varying complexity does not matter to the user as they can still use the application and look at previous results while the new analysis is being conducted. Researchers have attempted to overlay these analyses on real world objects using an augmented reality system as well [51]. Previous research into teaching structural analysis using FEA within virtual environments [4, 52, 53, 54] has been analyzed, and similar methods have been implemented into this work. This work aims to expand on the previous work from literature by directly implementing the structural analysis solver into the VR simulation for a real-time design environment. Within the implementation proposed in this work, actions should be taken to minimize latency, but a main question is at what complexity (total number of nodes), the performance reaches a choke-point and starts increasing latency. At that point, further actions should be taken to reduce processing demand and minimize virtual reality sickness, and properly follow industry best practices [32].

1.3 Thesis Summary

In this work, the implementation of a real time FEA solver within a virtual reality simulation will be discussed. The creation of the structure can either be done within the environment or loaded in beforehand. The user is then able to reconfigure the structure as well as adjust boundary conditions, material properties and forces all within the simulation. This structure can then be processed and analyzed by the solver and the deformed solution should be outputted back into the VR simulation. For low latency visualization in VR, this work will render elements of the

deformed structure as straight cylinders. This is an accurate portrayal for truss elements but is an oversimplified approach for visualizing frame elements and will need to be addressed in future work. The creation of this software will be investigated in both Unity and A-Frame Engines, and further teaching scenarios will also be developed as a result. The contents are organized as follows:

- Chapter 2 overviews the preliminary work done on this project leading up to the real-time design tool. It discusses various technologies explored and preliminary use cases that motivated this work. Observations from previous work is discussed here and the various lessons learned are defined.

- Chapter 3 describes the formulation of the elements used in finite element analysis and the implementation a generic FEA solver. This work will focus on using 3D frame elements specifically, but other elements such as truss and beams will be analyzed.

- Chapter 4 discusses the implementation of a finite element analysis solver within virtual environments and testing the accuracy of that analysis. The solver will be implemented in both A-Frame and Unity environments and user interaction capabilities will be created. The accuracy of the solvers will be tested against the ABAQUS commercial FEA software.

- Chapter 5 discusses the capabilities of this solver for educational and collaborational purposes. It will explore 3 different use case scenarios aimed at different audiences and the modifications made to improve the quality of the simulations.

- Chapter 6 summarizes the results of this work and discusses potential areas that could be researched to improve it further.

2. PRELIMINARY VIRTUAL REALITY DEVELOPMENTS FOR ENGINEERING

While the primary contributions of this thesis as outlined in subsequent chapters is in realtime design and analysis of structures, the approaches taken were all motivated by a variety of preliminary development tasks that were accomplished early in the development of this thesis that will be discussed in this chapter. Each software/scene/experience design task was undertaken with the twin goals of expanding the Virtual Reality engineering (especially mechanics) toolkit and increasing the understanding of the challenges, promises and pitfalls associated with these technologies. All avenues that will be discussed in this chapter produced important lessons that helped mold the Virtual Reality implementations addressed in Chapters 4 and 5. Prior technologies and workflows that were researched will be described here and important observations and lessons learned will be explicitly addressed.

2.1 Finite Element Post-Processing via the zSpace Platform

zSpace is a tabletop VR display which functions as a computer monitor that tracks user gaze through special sensors and glasses with a 6DOF stylus for user input. Studies showed that there were positive correlations between learning retention and hands-on Virtual Reality learning experiences using zSpace [55]. zSpace aims to capitalize on the educational market for VR hardware by providing a simpler alternative to VR headsets. Targeted at collaborative classroom experiences, they have seen seen relative success in adoption by K-12 classrooms across the US. Further, in 2015 the commercial finite element software ABAQUS was fervently sharing its newly developed potential for rendering its post-processed results on this platform. It was clear that custom tools might allow us to share this with aerospace engineering students at Texas A&M. Engineering students and researchers use ABAQUS as a tool to analyze the various structure that they design. From airplane wings and fuselages to discrete origami shape memory alloy structures. The idea was to understand whether a VR learning area could be created for students to collaborate in a similar fashion to figure 2.1 below.

Figure 2.1: Two students using a zSpace system [55]

VR Development Task(s)

zSpace was utilized in this work in the hopes of using their technology to easily post-process and visualize ABAQUS models. A partnership between zSpace and Dassault Systemes Simulia (the creators of ABAQUS), showed promising capabilities of a direct workflow to visualize FEA models in VR. Specifically, they were widely promoting their Living Heart Project (LHP) shown in Figure 2.2. LHP is a compelling experience that allowed users to analyze the internal geometry and slices of a beating heart [56] on the zSpace platform. Collaborating with Simulia engineers, python scripts were written that would output ABAQUS models and allow them to be visualized on the zSpace system. Rendering a complete model with cuts and over different output fields took a long time to process and resulted in many gigabytes of data. The visualization software was also not extensible and additional user interaction capabilities could not be developed.

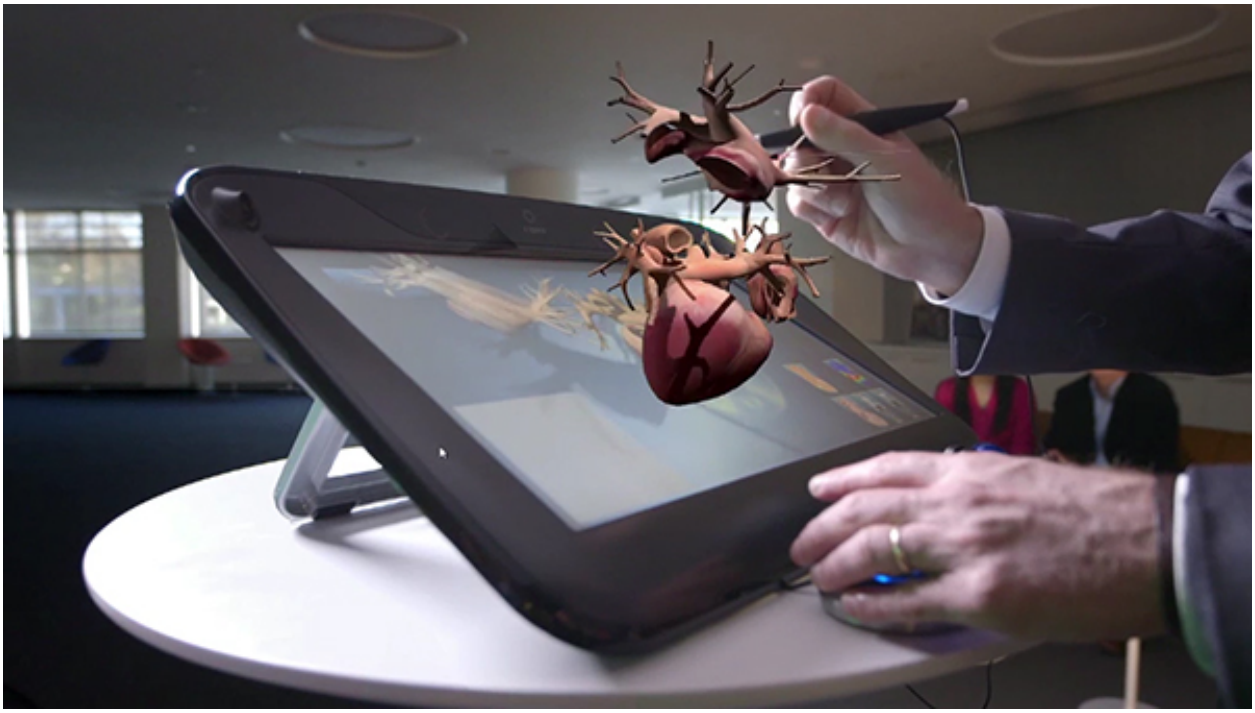


Figure 2.2: Living Heart Project on zSpace as developed by Simulia [48]. Various output fields can be rendered over the 3-D deforming model, including muscle strain and electric potential.

Observations

zSpace uses a shutter tracking system that updates the projection of the model on the screen to appear 3D in the user's eyes. This approach has multiple drawbacks as users will need to place themselves (i.e. their eyes) within a rather narrow angle from the center of the screen to be tracked by the system. As a result of this singular tracking and re-projection method for visualization, it is not truly multi-user. It is impossible to have the second user's head at the exact same location as the first user, so the projection from the zSpace will not be correct for a 3D visualization from the second user's perspective. This hurts zSpace's usability as a collaborative VR tool. Furthermore, the ability to directly visualize ABAQUS results on the zSpace in real-time had not been fully developed as discussed above. Results of the kind shown in Figure 2.2 were generated using a set of proprietary scripts developed by Simulia personnel. Generalizing these steps into a workflow that would be applicable to any analysis performed by students or researchers was difficult due to

the amount of prior knowledge needed. To use the zSpace for visualization, they would have to learn how to modify the python export process for their specific model and how to use the under-documented zSpace SDK. As a result, real-time rendering and post-processing of analysis was not as direct as was shown in the LHP example. Third party softwares, such as TechViz aimed to streamline the process and allow the direct use of ABAQUS in zSpace. However, these softwares cost upwards of $10,000 a year for a license and are still hindered by the capabilities of the zSpace system.

Lessons Learned

Visualizing ABAQUS models on the zSpace was a confusing and not very rewarding process. The amount of time spent visualizing the model pales when compared to the amount of time spent exporting the model for visualization. However, valuable lessons were learned in the process that sparked the motivation of a realtime design tool for FEA in virtual reality. The zSpace exporting process taught us that ABAQUS results could be exported into a Virtual Reality Media Language (VRML) format for use by another application. With some relatively straightforward processing of the VRML using Blender, it can be imported into a 3D simulation engines such as Unity to view. While this could have enabled further development of a tool for the zSpace platform, the zSpace Software Development Kit (SDK) lacks a development community and concrete implementation examples which hindered further work. This workflow to transfer ABAQUS results into an Unity environment later enabled the visualization of ABAQUS results on HMDs that provide a more realistic and immersive experience. This will be discussed further in Section 2.3.

2.2 VR Environment for Mechanical Design

The engineering design of 3D parts benefits from such softwares as SolidWorks and CATIA rendered onto 2D computer screens. While the mouse and keyboard provide great precision for drawing purposes, the 3D visualization of these designs suffer from projection and perspective bias. As we transitioned from the zSpace into much more widely acceptable and capable development environments, we explored the need to aid the mechanical design process by implementing virtual

reality for visualization.

VR Development Task(s)

The Unity simulation engine and various VR/AR headsets were utilized to view the designs. A workflow was developed to export full SolidWorks assemblies (e.g. many distinct parts precisely positioned relative to each other in a global coordinate system) as VRML files into the Blender 3D modeling software. In Blender, these models would be reconfigured to maintain the assembly hierarchy and to avoid negative 2D properties such as backface culling. Backface culling is a common shortcut method taken by visualization processes to render only the side visible to the user and not the backside aka backface. While this would be a great consideration for 2D screens, VR environments allow the user to navigate and manipulate objects, therefore making it necessary to render the backface. These updated models would then be exported as Filmbox (FBX) files for use in simulation engines. The Unity Engine was selected for this work because it has the largest developer community and very well documented SDKs. The headsets used were the Microsoft Hololens and the HTC Vive. The Hololens was initially used for Augmented Reality testing purposes, but was quickly abandoned as it relies on on-board computing and struggled to perform as more complex SolidWorks assemblies were introduced. Further, its limited field of view (35°) disallowed believable interaction with larger assemblies, including wings at 1:1 RC model scale as seen in Figure 2.3.

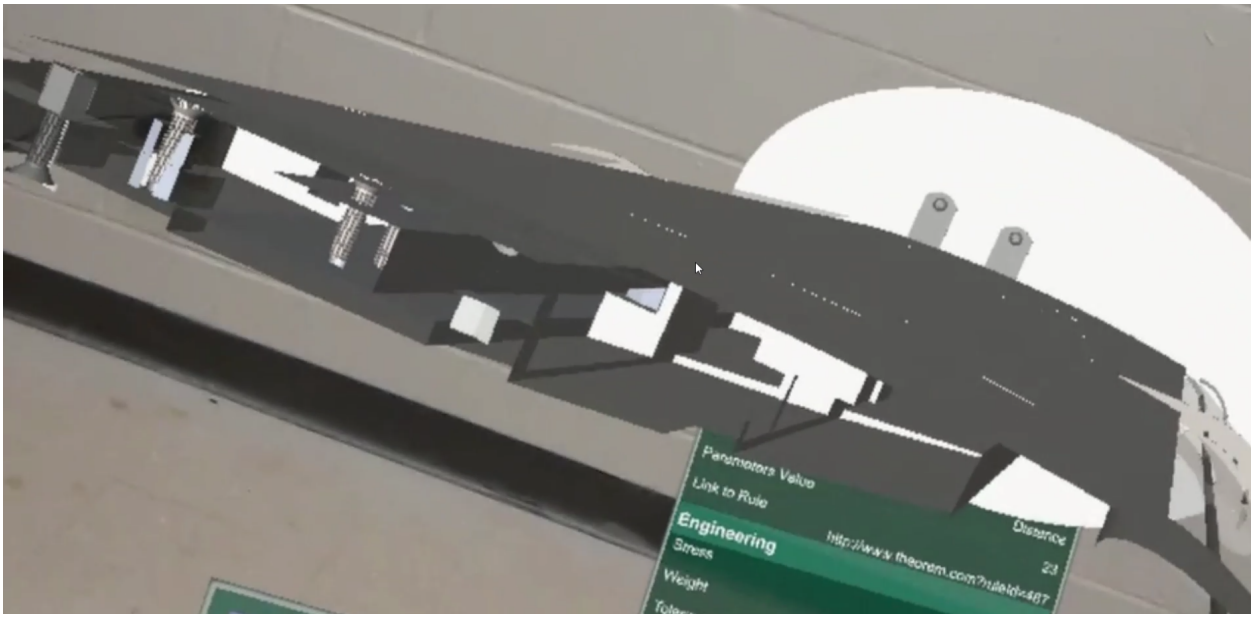


Figure 2.3: Auxiliary rendering of a SolidWorks designed wing viewed on Microsoft Hololens. In a users experience, the limited field of view prevented the entire wing from being viewed at once.

The HTC Vive was a more promising visualization device as the installed graphics card (Nvidia GTX 1080) on the host computer was leveraged to provide a higher fidelity simulation. Within the VR environment of the Vive, the user was able to move around the design and interact with each individual part using the controller, this is shown in Figure 2.4 below.

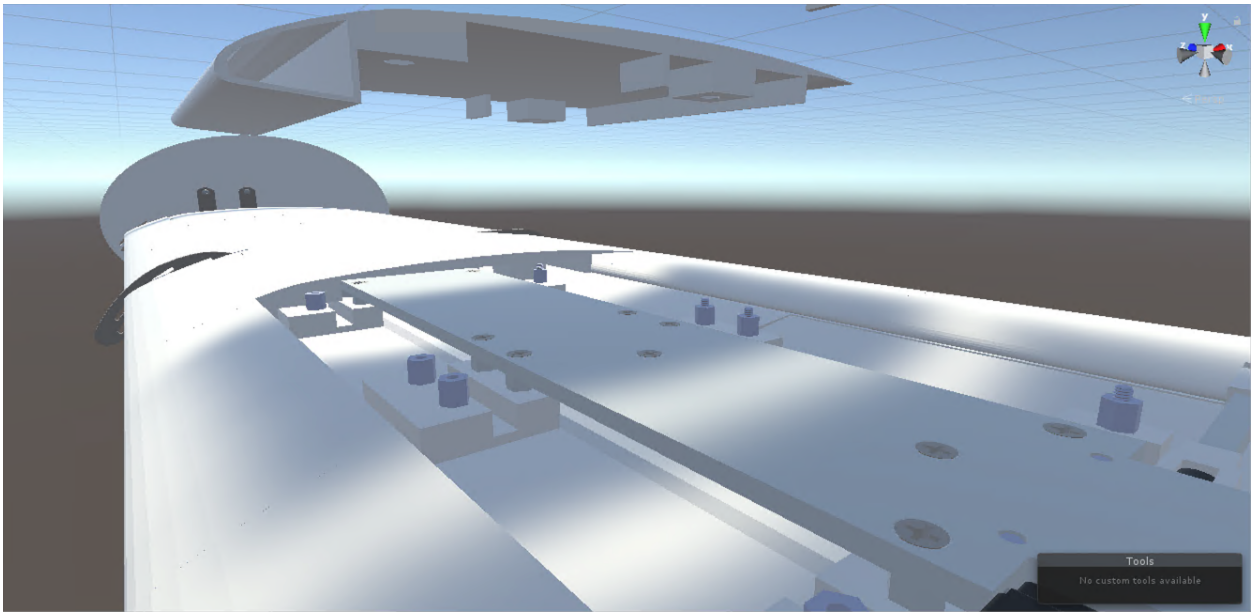


Figure 2.4: SolidWorks designed wing wind tunnel model (white 3-D printed plastic over grey aluminum structure) as viewed in Unity environment.

Observations

Viewing 3D models and disassembling them in a virtual reality environment showed promise as an engineering design visualization tool. Many students brought their own models to visualize within the Immersive Mechanics Lab and the simulation provided them with fresh insight that they would not have gained otherwise, such as on the aforementioned 2-D screens which SolidWorks and similiar models are usually rendered. The most memorable of these sessions involved the award-winning Texas A&M SAE Heavy Lift team. The design goal for their plane was to carry a large amount of tennis balls and conduct multiple objective-based test flights in rapid succession.

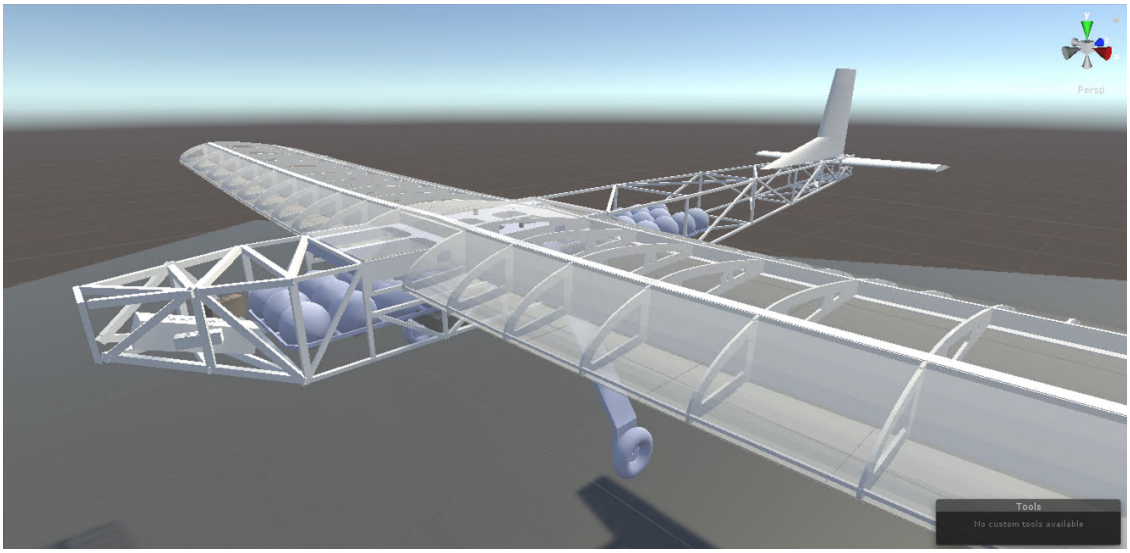


Figure 2.5: SAE heavy lift plane visualization; detailed assembly and even part transparency are clearly rendered.

The SAE team were preparing for a national competition and their design had already passed the preliminary design review. Immediately upon transferring into the VR simulation, they noticed a critical error. One design requirement was that spent batteries be quickly swapped for fresh batteries between test flights. However, their design was not optimized for this functionality as structural members blocked all access to the battery. Removing critical structures between flights would be difficult and would weaken the overall structural integrity of the plane. As a result of the VR simulation and a relatively brief immersive experience, the team decided to redesign the battery cover area to be more easily accessible.

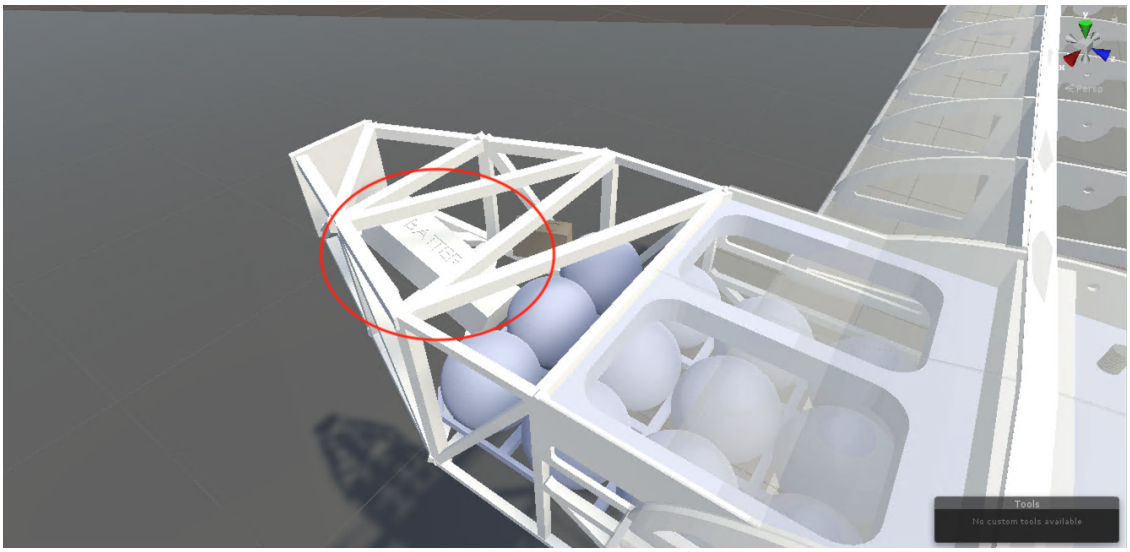


Figure 2.6: SAE Heavy Lift battery area found to be poorly designed using traditional 2-D tools but quickly discovered during VR design immersion.

Lessons Learned

From the many collaborative experiences with student designs, important insights were gleaned that could be useful when designing the simulations and scenes that comprise the primary contributions of this work.

- User interaction is valuable and users enjoy being able to use their hands/controllers to manipulate objects.
- Scene design provides the users with a sense of presence and should not be overlooked.
- Augmented Reality experiences on the Microsoft Hololens are not immersive and have many limitations. Further developments on that platform were suspended in lieu of focused VR tool design and deployment.
- Students and/or engineers can have moments of surprising insight when they are immersed in "worlds" containing complex bodies and assemblies for which they are responsible and which they may have configured.

2.3 Immersive Finite Element Post-processing in a VR Environment

As in Section 2.1, we were motivated to continue to explore methods by which the diversity of finite elements (and even computational fluid dynamics results) generated through the aerospace engineering field could be experienced in a deeper way, as from the *inside*. Given the failure of the zSpace and Hololens devices and the successes of SolidWorks model visualization on the HTC Vive; it was clear that virtual reality offered a much better possible solution, so an effort to design, visualize, and test custom tools towards this end was initiated.

Aerospace Structures Design (AERO 405) is a structural optimization course at Texas A&M that teaches senior aerospace engineering students about structural analysis and design optimization algorithms. The final project for the course asks teams of students to design and optimize a structural model of their choosing. Many students decide to design wings for their capstone teams, while other projects revolve around bicycle wheels, shape memory alloy devices, surfboards, etc. The goals of the project were to reinforce the structural analysis topics that had been taught throughout their undergraduate careers, and to make informed design trade studies that a professional structural analyst would make. These designs are made using the ABAQUS FEA software and students spend many hours analyzing these models on their computer screens. As a means of testing our newly developed VR tools, an extra credit opportunity was introduced in this course for students to transfer their ABAQUS results to the new immersive FEA scenes and assess their designs there.

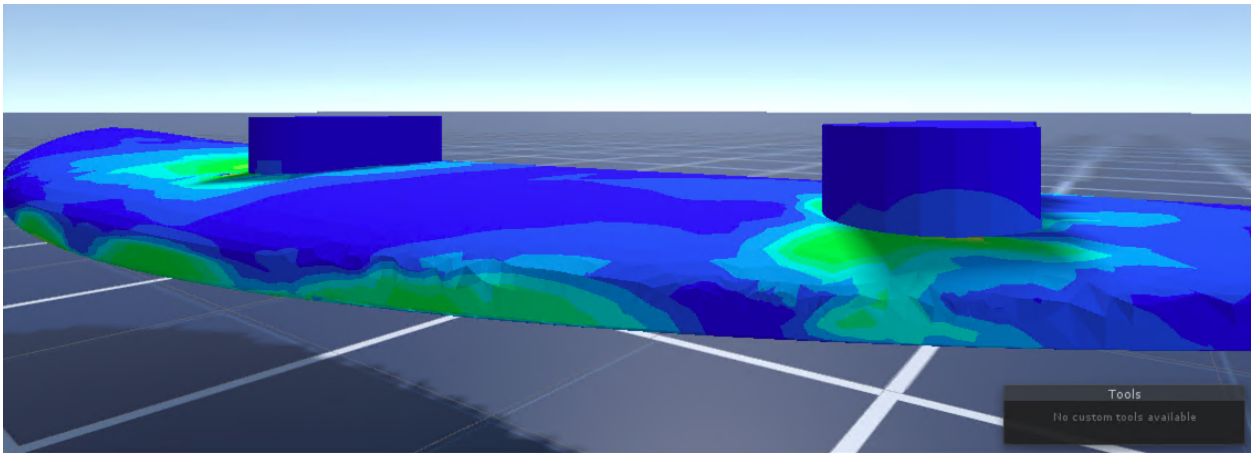


Figure 2.7: ABAQUS design of a surfboard under loading; the stress contours from the force exerted on the board by the surfers feet are visualized here.

VR Development Task(s)

The primary output of ABAQUS FEA are odb (output database) files, which are primarily used to view contours representing various fields (e.g. temperature, stress, deformation) mapped as textures onto 3-D models, often in their deformed state. In exploring the zSpace platform, it was discovered that the textured 3-D bodies could be exported as VRML files. The SolidWorks workflow from section 2.2 that used Blender to convert a VRML model into a FBX file for Unity was employed on the ABAQUS models as well. This allowed the ABAQUS model geometry and stress contours to easily be visualized in the immersive VR environment of the HTC Vive. Since the Vive employs room-scale tracking of the headset and controllers, the students could physically walk around their designs and manipulate them using the controllers. A workflow was later implemented to view cuts of the model across X-Y-Z planes in the hopes of gleaning deeper insight. However, this was later removed because it is a tedious process that required a large amount of data and could easily be replicated by the user moving their head inside the model.

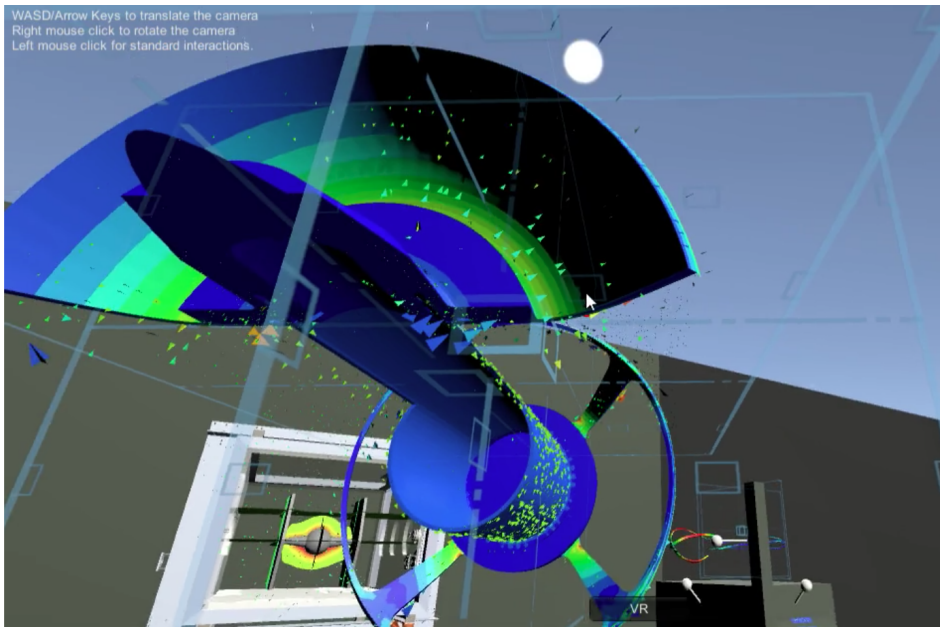


Figure 2.8: Internal cut view of an ABAQUS Engine

Observations

Many students were able to use the VR simulation to see stress concentrations and future areas of interest that they might have otherwise missed. If fact, a requirement of the instructor (D. Hartl) is that students use the VR experience to explain some interesting or critical feature of their design to him. A very interesting case emerged when a team saw that their final model did not meet the design criteria they desired. This team was designing an airplane wing to minimize tip twist, however, their code was quantifying tip twist incorrectly, and this had not been recognized when exploring the model on 2D computer screens. When the team was immersed in the VR experience with their model for only a few minutes, they quickly recognized the unacceptably large wing twist. VR enabled the users to gain a more critical perspective of their own designs and many teams went on to implement their insights as changes to the final design.

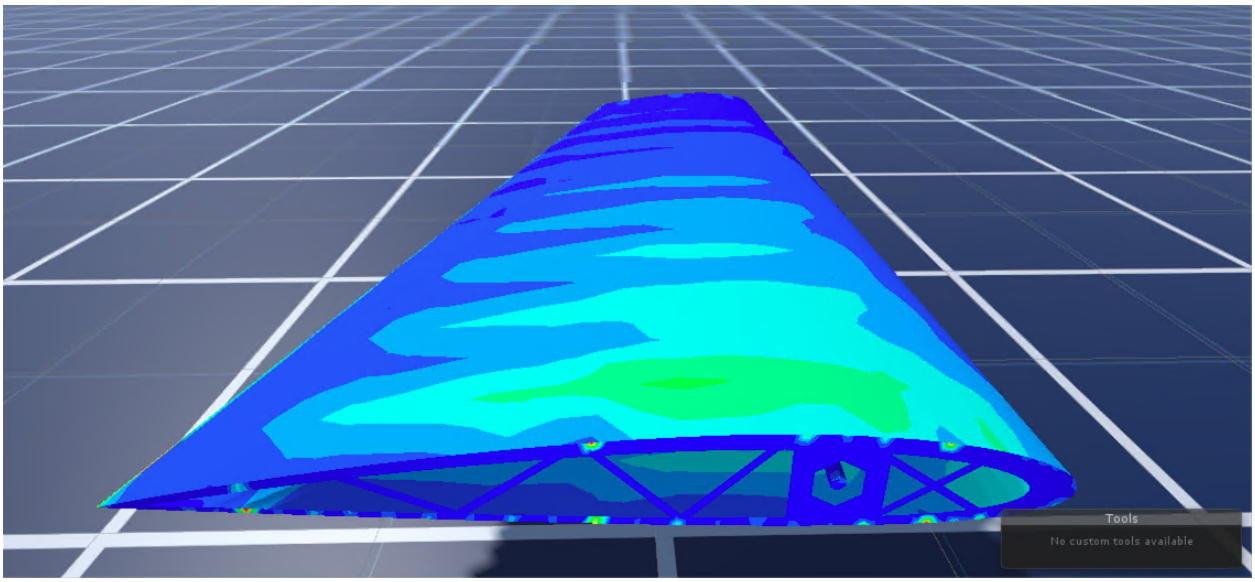


Figure 2.9: View of an ABAQUS wing after a redesign made necessary after students noticed a design flaw in VR that had gone unnoticed during many hours of development on conventional 2-D screens.

Lessons Learned

User feedback demonstrated the need for being able to make modifications to their designs from within the VR simulation itself. A workflow for a simple way to achieve that does not currently exist. This drove the motivation for developing a realtime FEA tool which is the basis for this work. A few important insight gleaned are that:

- Users benefited from and even enjoyed being scaled smaller than their designs so that they could move through them and visualize from both the inside and the outside.
- Users preferred leaving the model static and moving their own bodies instead of moving the model.
- Users appreciated being able to visualize stress contours on their model; it is important that a distinguishable color scheme for stress should be implemented in this work.
- Users enjoyed room-scale physical navigation by walking, and sometimes by crawling. The

immersive environment made exploring various design features by crawling such a common occurrence that the lab eventually had to get the floors carpeted.

2.4 EducatAR: Supplementing The Stem Classroom Experience

Since the FEA visualization efforts using Unity had been successful and student feedback had been almost entirely postive, we felt that we should explore other VR platforms for delivering this and similar content, but in a shared way. Further, student response inspired the exploration of methods for delivering content from other technical disciplines (e.g. atmospheric science, materials science), and a team was assembled to which I was the primary technical contributor.

Texas A&M Triads for Transformation , also known as T^3, is a seed grant for funding collaborative research at Texas A&M University. Each round, 100 projects with goals of advancing transformational learning, increasing multidisciplinary collaboration, and moving ideas from concepts to creations were selected. Each project consisted of 3 professors from different disciplines at Texas A&M working together on a shared vision for their project. The EducatAR project was proposed by a team led by Dr. Darren Hartl and also included Dr. Chris Nowotarski (Atmospheric science) and Dr. Raymundo Arroyave (Materials science). T^3 funded them to research how the STEM classroom experience could be enhanced through the use of virtual reality technologies. They proposed the creation of VR applications in the areas of computational mechanics, materials science and atmospheric science in order to make education in those disciplines accessible to a wider variety of learning styles.

VR Development Task(s)

Traditional VR headsets (e.g. HTC Vive) represent high-end hardware that come at a high cost. To make VR accessible to every student required the use of low-cost VR devices like the Google Cardboard shown in Figure 2.10. This headset costs about $5 and is configured so that the user can slip a smartphone of any size inside its sleeve to create the VR experience. The downside to this headset is the lack of control as it can only track the users head rotations via smartphone accelerometers and has only one button for actions. While an application could be developed for

this using the Unity engine, this would not be a collaborative experience so the A-Frame platform was utilized instead. A-Frame hosts the VR experience on a web server (e.g. in the cloud) so any user with a phone, computer, or VR headset can access it. This device agnostic approach allows users to be networked into the same experience, simulating the feeling of a collaborative classroom. A-Frame is programmed using the $HTML$ language to handle all the on-screen visualizations, and the $JavaScript$ language to handle all the simulation back end (e.g. physics interaction, server connection, etc). The device agnostic approach meant that interactions could be programmed generally, and all devices would have similar views and functionality in the VR environment. However, a noticeable weakness to this is that specific user interaction capabilities are extremely difficult to program in this generic manner.

Figure 2.10: Google cardboard VR device

Observations

Many scenes were created to show different aspects of the possibilities with A-Frame for education. Figure 2.11 below shows one frame from an animated A-Frame experience where students can fly through isosurfaces (e.g. velocity magnitude) compiled from weather system models applied to atmospheric data at different time points in a storm. They can virtually experience the data

from the beginning, middle, and end of a storm in order to gain a deeper understanding of how weather patterns are formed and how they evolve.

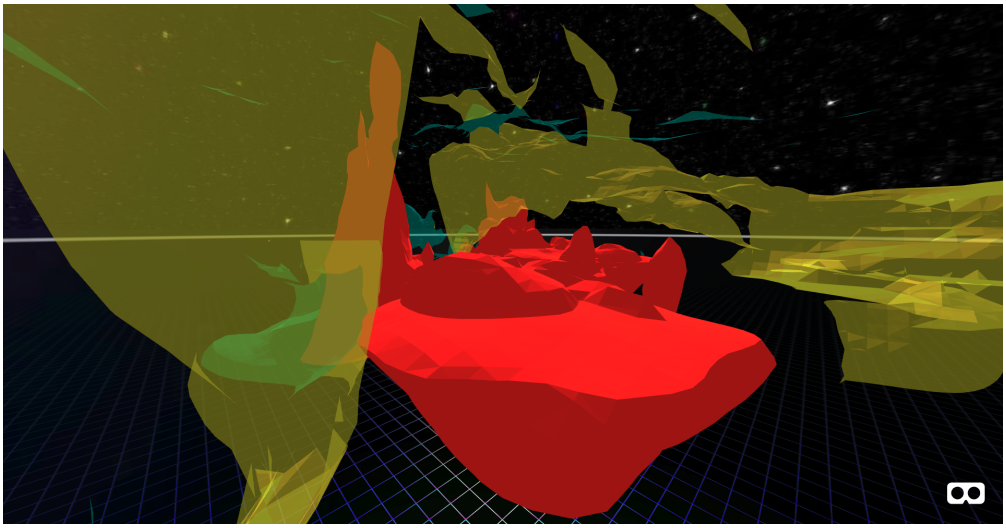


Figure 2.11: T^3 Atmospheric science experience in A-Frame allowing students to immerse themselves in an storm model as it evolves

Figure 2.12 shows the inside of a computed structure in a VR environment based on the thermodynamically stable growth of precipitates during a heat treatment process as applied to metals. This A-Frame simulation allows the users to move through the internal volume of a specimen and make observations and analytical notes about it. This experience was utilized with many different types of specimen for the user to choose between and navigate through. Animations were made so that the user could see specimen particles at different stages of growth. This same workflow was later utilized by the Computational Materials Science (CMS^3) Summer School to provide materials science graduate students with an immersive technique to visualize and investigate materials phenomena across various scales.

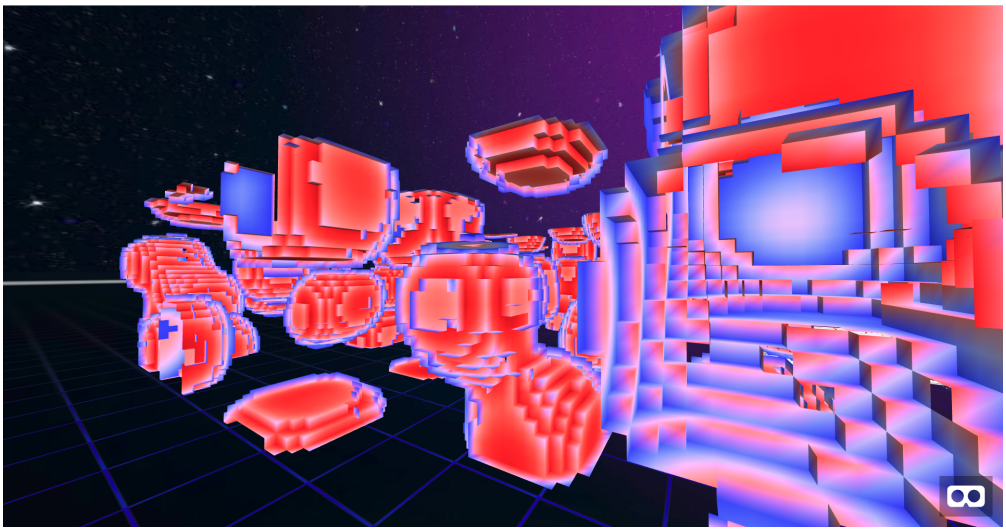


Figure 2.12: T^3 Materials science experience in A-Frame allowing students to visualize and navigate around the internal precipitate growth of a computed structure

Figure 2.13 shows ABAQUS results in a similar fashion to that already discussed in Section 2.3, but employing webVR so that results could be viewed anywhere in the world via a cardboard device (Figure 2.10). With this workflow, students can upload their own designs to a A-Frame environment to visualize on their phones and cardboard devices. This enables a collaborative educational environment as students can remotely view and discuss the ABAQUS model for their project.

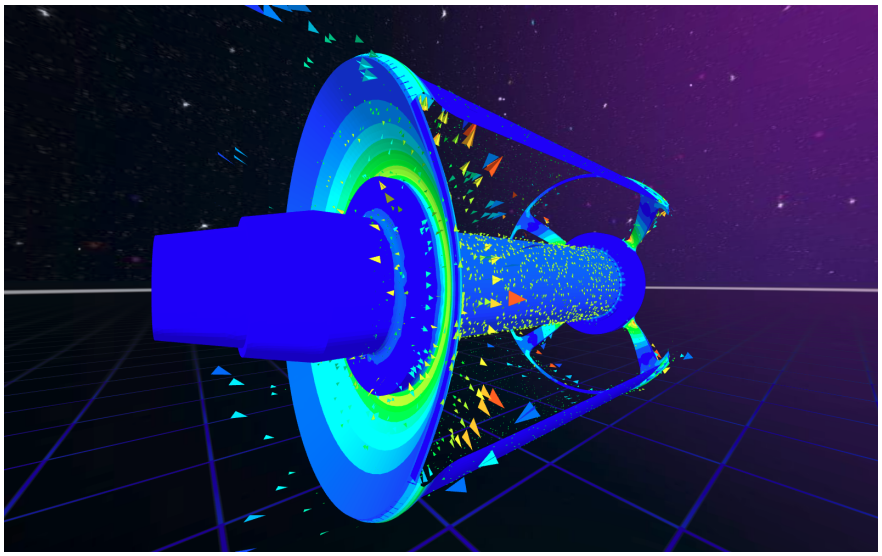


Figure 2.13: T^3 Computational Mechanics experience in A-Frame showing the results of FEA analysis conducted on an engine shell.

Lessons Learned

With the T^3 research project, insight was gleaned into how successful A-Frame applications could be developed in the future. These lessons learned are:

- The need for a on-screen user interface: Users complained that the one button on the cardboard device did not allow for enough user interaction capabilities.

- When the model to be visualized was complex/large, longer load times were required on mobile devices and the rendering of even simple motions could be performed only with very high latency. Further investigation showed that A-Frame uses the processor of the device to parse the web-page and perform all graphical operations so most handheld devices do not have near the requisite computing power. Therefore, models rendered in such environments would need to be relatively small.

- Users discussed preferring to navigate the A-Frame simulation through pre-defined teleportation points over "flying" through the simulation on the cardboard device. Slowing the speed of flight made that technique more usable but teleportation provided a more controlled and direct navigational experience.

3. THEORY OF FINITE ELEMENT ANALYSIS

Finite element analysis (FEA) is the simulation and analysis of physical phenomenon through the use of the finite element method (FEM). While FEM as a concept is multi-disciplinary, in terms of structural analysis, this methodology refers to the concept of subdividing a larger more complex structure into smaller parts called finite elements. These finite elements are analyzed within a local coordinate system first and then adjusted and assembled to fit into the global context of the structure. The conducting of FEA is an important step in the structural design process as it helps the engineers model and predict the feasibility of their designs. Depending on the dimensionality and fidelity of the analysis, different types of elements, such as bar, beam, and frame elements are used. The mathematical formulation of each element type is dependent on both the number of degrees of freedom (DOF) per node and number of nodes within an element. This work shall only consider elements with 2 nodes such as bars, beams, and frames.

This work mainly utilized 3D frame elements for its analysis, but first, bar and beam element formulation in local 2-D space are explained as they are a fundamental prerequisite that will be built upon [57, 58, 59, 60, 61]. This chapter will culminate with the explanation and implementation of a generic FEA solver that solves 3D frame structures of any arbitrary size.

3.1 Bar Element

Bar elements are a simple type of finite element with one degree of freedom at each node for axial deformation. Thus, the element can only undergo tensile or compressive loading, as shown in Figure 3.1. In a structure consisting of multiple bar elements, the individual members are joined together by pins or hinges, so that only forces are transmitted across the elements.

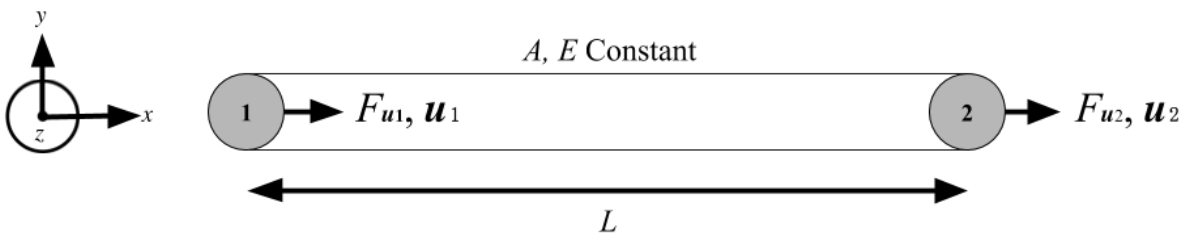


Figure 3.1: Bar Element subjected to axial force

When analyzing a single bar element inside a local coordinate system where x=0 coincides with node 1, the displacement of that element can be written in the form:

$$\boldsymbol{u} = \boldsymbol{N}\boldsymbol{d_e} \tag{3.1}$$

In this equation, $\boldsymbol{u}$ approximates the axial displacement of the element, $\boldsymbol{N}$ represents the matrix of shape functions that describe the element, and $\boldsymbol{d_e}$ is a vector of the actual displacement at the two nodes of the element. $\boldsymbol{d_e}$ is broken down into the two axial displacements at each node as:

$$\boldsymbol{d_e} = \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \tag{3.2}$$

The general form for axial displacement in a bar element can be written as:

$$\boldsymbol{u} = \alpha_0 + \alpha_1 x = \begin{Bmatrix} 1 & x \end{Bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \end{Bmatrix} = \boldsymbol{p^T}\boldsymbol{\alpha} \tag{3.3}$$

In 3.3 above, $\boldsymbol{\alpha}$ is a vector that represents two unknown constants (α_0, α_1), and $\boldsymbol{p^T}$ is a vector of polynomial basis functions. In deriving the matrix of shape functions, $\boldsymbol{N}$, the length of the element is represented as L, thus in the local coordinate system at $x = 0, u(0) = u_1$, and at $x = L, u(L) = u_2$. This gives the following system of two equations created by 3.3 at x=0 and

x=L:

$$\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & L \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \end{Bmatrix} \tag{3.4}$$

Solving 3.4 for $\boldsymbol{\alpha}$ gets the equation in the form:

$$\begin{Bmatrix} \alpha_0 \\ \alpha_1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{-1}{L} & \frac{1}{L} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \tag{3.5}$$

Substituting 3.5 above into 3.3 below then obtains

$$\boldsymbol{u} = \boldsymbol{p^T \alpha} = \begin{bmatrix} 1 & 0 \\ \frac{-1}{L} & \frac{1}{L} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} 1 - \frac{x}{L} & \frac{x}{L} \end{Bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \boldsymbol{N d_e} \tag{3.6}$$

$\boldsymbol{N}$ is now a matrix of shape functions, described as:

$$\boldsymbol{N} = \begin{bmatrix} N_1 & N_2 \end{bmatrix} \tag{3.7}$$

where:

$$\begin{aligned} N_1 &= 1 - \frac{x}{L} \\ N_2 &= \frac{x}{L} \end{aligned} \tag{3.8}$$

Since a bar element in 2D is very simple, the displacement within the element only varies linearly. This can be seen when the shape functions are explicitly substituted back into the displacement function to get:

$$\boldsymbol{u} = N_1 u_1 + N_2 u_2 = u_1 + \frac{u_2 - u_1}{L} x \tag{3.9}$$

Now that the displacement of the bar element is defined mathematically, this can be used to under-

stand how the stress and strain are defined within the element. Strain ($\boldsymbol{\varepsilon}$) is defined as:

$$\varepsilon_x = \frac{\partial u}{\partial x} = \frac{u_2 - u_1}{L} \tag{3.10}$$

This can be alternatively written in matrix form as:

$$\varepsilon_x = \frac{\partial u}{\partial x} = L\boldsymbol{N}\boldsymbol{d_e} = \boldsymbol{B}\boldsymbol{d_e} \tag{3.11}$$

where L is the differential operator, and $\boldsymbol{B}$ is the strain matrix represented as

$$\boldsymbol{B} = \ L\boldsymbol{N} = \frac{\partial}{\partial x}\begin{bmatrix} 1 - \frac{x}{L} & \frac{x}{L} \end{bmatrix} = \begin{bmatrix} \frac{-1}{L} & \frac{1}{L} \end{bmatrix} \tag{3.12}$$

Once the strain matrix $\boldsymbol{B}$ is obtained, the stiffness matrix $\boldsymbol{k_e}$ of the bar element in the local coordinate system can be obtained.

$$\boldsymbol{k_e} = \int_{V_e} \boldsymbol{B^T c B}\, dV \tag{3.13}$$

In 3.13 above, the strain matrix is integrated across the volume of the element V_e , and $\boldsymbol{c}$ is a material constant matrix. For a bar element $\boldsymbol{c}$ is simply the elastic modulus E of the material. Expanding 3.14, this gives:

$$\boldsymbol{k_e} = A_e \int_0^L \begin{bmatrix} \frac{-1}{L} \\ \\ \frac{1}{L} \end{bmatrix} \ E \ \begin{bmatrix} \frac{-1}{L} & \frac{1}{L} \end{bmatrix} \, dX \tag{3.14}$$

Solving 3.14 above, a simplified stiffness matrix shown below is obtained where A_e represents the cross-sectional area of the bar element.

$$\boldsymbol{k_e} = \frac{EA_e}{L} \begin{bmatrix} 1 & -1 \\ \\ -1 & 1 \end{bmatrix} \tag{3.15}$$

This shows how a single bar element is formulated. These bar elements are in the local coordinate

system and will need to be transformed into the global coordinate system before the analysis can be conducted. To solve for the actual displacement, $\boldsymbol{D_g}$ of the structure consisting of bar elements, the fundamental system of equations shown in equation 3.16 for the finite element model must be used.

$$\boldsymbol{F_g} = \boldsymbol{K_g D_g} \tag{3.16}$$

$\boldsymbol{f_e}$ is the vector that contains values for the forces applied at the nodes for this element. In this bar element formulation with only axial degrees of freedom, only F_{u_1} and F_{u_2} are allowed to be nonzero.

This section covered the basics of how to formulate the stiffness matrix for a single bar element in the local coordinate system. Planar truss elements are an extension of the bar element where it is rotated into 2D space so it has two degrees of freedom and can displace in both the X and Y directions. This will not be discussed in this section because the end goal was to show how frame elements are comprised of both bar and beam elements. To utilize this bar element in the derivation of a frame element formulation, the bar element is mathematically expressed in a 6 DOF configuration by including transverse (v) and rotational displacement (θ) at the nodes. The resulting displacement vector is

$$\boldsymbol{d_e} = \begin{Bmatrix} u_1 \\ v_1 \\ \theta_{z_1} \\ u_2 \\ v_2 \\ \theta_{z_2} \end{Bmatrix} \tag{3.17}$$

However, since a bar is only able to axially deform, these additional displacements are set to zero. Thus when constructing the local element stiffness matrix in a 6 DOF configuration, rows and columns corresponding to the transverse and rotational displacements are also set to zero as

shown:

$$\boldsymbol{k_e} = \begin{array}{c} \begin{matrix} u_1 & v_1 & \theta_{z_1} & u_2 & v_2 & \theta_{z_2} \end{matrix} \\ \begin{bmatrix} \frac{EA}{L} & 0 & 0 & \frac{-EA}{L} & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 \\ & & 0 & 0 & 0 & 0 \\ & & & \frac{EA}{L} & 0 & 0 \\ & symm. & & & 0 & 0 \\ & & & & & 0 \end{bmatrix} \end{array} \tag{3.18}$$

3.2 Beam Element

2D Beam elements are a type of finite element that is more commonly used than bar elements. Like bar elements, it is a geometrically straight bar of an arbitrary cross section; however as shown in Figure 3.2, beams only deform in directions perpendicular to its axis (x-axis in this case). Beam elements have two degrees of freedom at each node, which allows the element to deform both transversely (v) and rotationally (θ_z).

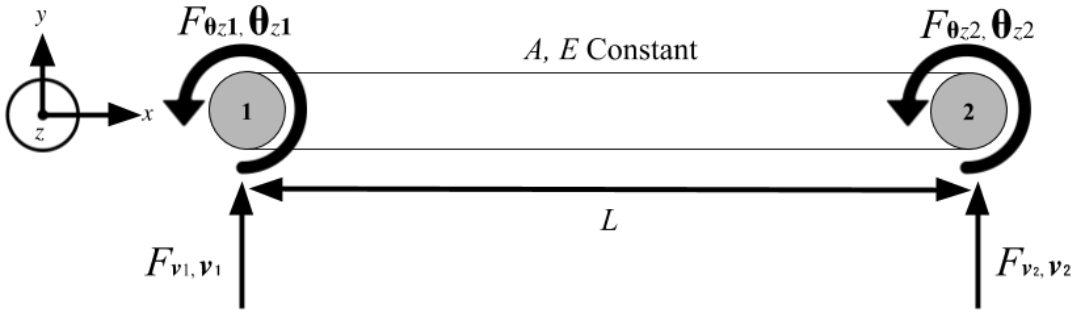


Figure 3.2: Beam element with positive modal displacements, rotations, forces and moments

In structures comprised of multiple beam elements, it is assumed that each element is joined together by welding; a method that enables both forces and moments to be transmitted between the beams. The formulation for beam elements used in this is derived from the Euler-Bernoulli

Beam Theory [62]. When calculating the displacement of a beam element, the equation is similar to equation 3.3 above, but with slight modifications. For a beam element of length L, assume that the transverse displacement function υ is:

$$\upsilon = d_1 x^3 + d_2 x^2 + d_3 x + d_4 \tag{3.19}$$

where d_1 through d_4. This is a third degree polynomial due to the 4 degrees of freedom in this element. Following Euler-Bernoulli beam theory, the boundary conditions are:

$$\begin{aligned} &\text{At Node 1}, x = 0: \\ &(1)\ \upsilon(x=0) = \upsilon_1 \\ &(2)\ \left.\frac{d\upsilon}{dx}\right|_{x=0} = \theta_{z_1} \end{aligned} \tag{3.20}$$

$$\begin{aligned} &\text{At Node 2}, x = L: \\ &(3)\ \upsilon(x=L) = \upsilon_2 \\ &(4)\ \left.\frac{d\upsilon}{dx}\right|_{x=L} = \theta_{z_2} \end{aligned} \tag{3.21}$$

Applying these boundary conditions and solving for the unknown coefficients gives:

$$\begin{aligned} &\upsilon(0) = \upsilon_1 = d_4 \\ &\upsilon(L) = \upsilon_2 = d_1 x^3 + d_2 x^2 + d_3 x + d_4 \\ &\frac{d\upsilon(0)}{dx} = \theta_{z_1} = d_3 \\ &\frac{d\upsilon(L)}{dx} = \theta_{z_1} = 3d_1 L^2 + 2d_2 L + d_3 \end{aligned} \tag{3.22}$$

Solving these equations for d_1, d_2, d_3, and d_4 gives:

$$\upsilon = [\frac{2}{L^3}(\upsilon_1 - \upsilon_2) + \frac{1}{L^2}(\theta_{z_1} + \theta_{z_2})]x^3 + [-\frac{3}{L^2}(\upsilon_1 - \upsilon_2) - \frac{1}{L}(\theta_{z_1} + \theta_{z_2})]x^2 + \theta_{z_1} x + \upsilon_1 \tag{3.23}$$

This can be written in matrix form as:

$$\boldsymbol{v} = \boldsymbol{N}\boldsymbol{d_e} \tag{3.24}$$

The displacement vector ($\boldsymbol{d_e}$) is defined as:

$$\boldsymbol{d_e} = \begin{Bmatrix} d_4 \\ d_3 \\ d_2 \\ d_1 \end{Bmatrix} = \begin{Bmatrix} v_1 \\ \theta_{z_1} \\ v_2 \\ \theta_{z_2} \end{Bmatrix} \tag{3.25}$$

The matrix of shape functions ($\boldsymbol{N}$) is defined as:

$$\boldsymbol{N} = \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix} \tag{3.26}$$

where the individual components are:

$$\begin{aligned} N_1 &= \frac{1}{L^3}(2x^3 - 3x^2L + L^3) \\ N_2 &= \frac{1}{L^3}(x^3L - 2x^2L^2 + xL^3) \\ N_3 &= \frac{1}{L^3}(-2x^3 + 3x^2L) \\ N_4 &= \frac{1}{L^3}(x^3L - x^2L^2) \end{aligned} \tag{3.27}$$

One of the basic assumptions made in simple beam theory is that planes remain planar after deformation. As a result, moments ($M(x)$) and shear ($V(x)$) are related to the transverse displacement as:

$$\begin{aligned} M(x) &= EI_z(\frac{d^2v}{dx^2}) \\ V(x) &= EI_z(\frac{d^3v}{dx^3}) \end{aligned} \tag{3.28}$$

Using the beam theory sign conventions and the above equations, the forces applied to the element

can be found by the following:

$$
\begin{aligned}
f_{\upsilon 1} &= V = EI\frac{d^3\upsilon}{dx^3}\bigg|_{x=0} = \frac{EI}{L^3}(12\upsilon_1 + 6L\theta_{z_1} - 12\upsilon_2 + 6L\theta_{z_2}) \\
f_{\upsilon 2} &= -V = -EI\frac{d^3\upsilon}{dx^3}\bigg|_{x=L} = \frac{EI}{L^3}(-12\upsilon_1 - 6L\theta_{z_1} + 12\upsilon_2 - 6L\theta_{z_2}) \\
f_{\theta_{z_1}} &= -M = -EI\frac{d^2\upsilon}{dx^2}\bigg|_{x=0} = \frac{EI}{L^3}(6L\upsilon_1 + 4L^2\theta_{z_1} - 6L\upsilon_2 + 2L^2\theta_{z_2}) \\
f_{\theta_{z_2}} &= M = EI\frac{d^2\upsilon}{dx^2}\bigg|_{x=L} = \frac{EI}{L^3}(6L\upsilon_1 + 2L^2\theta_{z_1} - 6L\upsilon_2 + 4L^2\theta_{z_2})
\end{aligned}
\tag{3.29}
$$

Recalling 3.16 from above will allow the definition of the element stiffness matrix for a beam element.

$$
\boldsymbol{f_e} = \boldsymbol{k_e}\boldsymbol{d_e} = \begin{Bmatrix} f_{\upsilon 1} \\ f_{\theta_{z_1}} \\ f_{\upsilon 2} \\ f_{\theta_{z_1}} \end{Bmatrix} = \frac{EI_z}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix}\begin{Bmatrix} v_1 \\ \theta_{z_1} \\ v_2 \\ \theta_{z_2} \end{Bmatrix} \tag{3.30}
$$

The element stiffness matrix ($\boldsymbol{k_e}$) can be individually defined as:

$$
\boldsymbol{k_e} = \begin{bmatrix} \frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} & -\frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} \\ & \frac{4EI_z}{L} & -\frac{6EI_z}{L^2} & \frac{2EI_z}{L} \\ & & \frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} \\ & & & \frac{4EI_z}{L} \end{bmatrix} \tag{3.31}
$$

In a similar manner to the bar element formulation, the stiffness matrix formulation for a beam element in the local coordinate system is be written below in 6 degree of freedom terms in order to aid with the next section where the stiffness matrix for 2-D frame elements is formulated. Since beam elements do not displace axially (u), the 1st and 4th columns in the matrix below attributed

to that are full of zeros.

$$\boldsymbol{k_e} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ & \frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} \\ & & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & \frac{2EI_z}{L} \\ & & & 0 & 0 & 0 \\ & symm. & & & \frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} \\ & & & & & \frac{4EI_z}{L} \end{bmatrix} \tag{3.32}$$

3.3 Frame Element

A frame element is formulated to be a straight bar of any arbitrary cross-section that combines properties of bar and beam elements, and is capable of accounting for both translational and rotational forces.

3.3.1 2D Frame Element

A 2D frame element has 3 DOF at each node (for a total of 6 DOF): axial displacement (u), transverse displacement (v), and rotation about the z-axis (θ_z). Each node is subjected to axial (F_x) and transvere (F_y) forces and bending moments ($F_{\theta z}$). This is depicted in Figure 3.5. For a frame

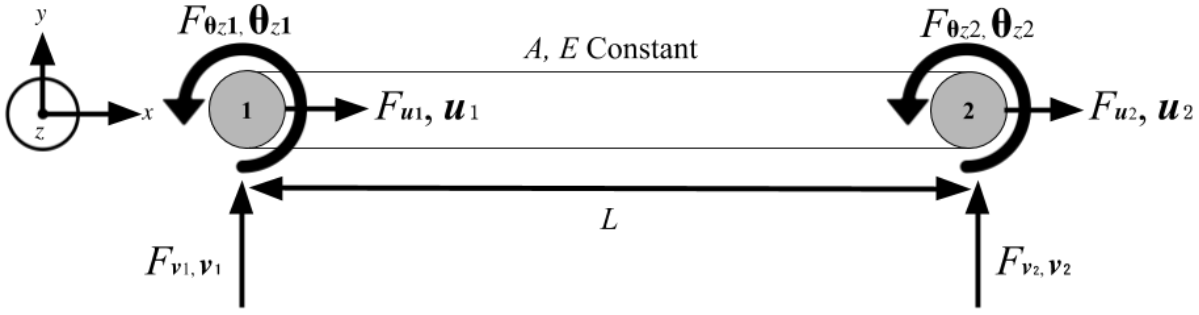


Figure 3.3: Example of a 2D frame element with three degrees of freedom per node (two translational, one rotational)

element, the displacement vector ($\boldsymbol{d_e}$) and force vector($\boldsymbol{f_e}$) can be written as seen in Equations 3.33 and 3.34 below.

$$\boldsymbol{d_e} = \begin{Bmatrix} u_1 & v_1 & \theta_{z_1} & u_2 & v_2 & \theta_{z_2} \end{Bmatrix}^T \tag{3.33}$$

$$\boldsymbol{f_e} = \begin{Bmatrix} F_{u_1} & F_{v_1} & F_{\theta_{z_1}} & F_{u_2} & F_{v_2} & F_{\theta_{z_2}} \end{Bmatrix}^T \tag{3.34}$$

The frame element stiffness matrix ($\boldsymbol{k_e}$) is formulated by combining the stiffness matrices from the bar element (3.18) and the beam element (3.32):

$$\boldsymbol{k_e} = \begin{array}{c} \begin{matrix} u_1 & v_1 & \theta_{z_1} & u_2 & v_2 & \theta_{z_2} \end{matrix} \\ \begin{bmatrix} \frac{EA}{L} & 0 & 0 & \frac{-EA}{L} & 0 & 0 \\ & \frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} \\ & & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & \frac{2EI_z}{L} \\ & & & \frac{EA}{L} & 0 & 0 \\ & symm. & & & \frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} \\ & & & & & \frac{4EI_z}{L} \end{bmatrix} \end{array} \tag{3.35}$$

Most systems analyzed by the finite element method are composed of many individual elements, each with their own local coordinate system. In order to create a more complex system composed of many elements, a global coordinate system will need to be employed to keep track of the location and orientation of each element. Nodes 1 and 2 for the element have now been renamed to corresponding global nodes i and j. In the local coordinate system of $x - y - z$, local nodes 1 and 2 have u and v translational components as well as a θ_z rotational component. These local displacements correspond exactly with the global coordinate system of $X - Y - Z$ with nodes i and j having D_{3i-2}, D_{3j-2} and D_{3i-1}, D_{3j-1} translational components and D_{3i}, D_{3j} rotational

components. The displacement vector for the element in the global coordinate system (D_e) is:

$$\boldsymbol{D_e} = \begin{Bmatrix} D_{3i-2} \\ D_{3i-1} \\ D_{3i} \\ D_{3j-2} \\ D_{3j-1} \\ D_{3j} \end{Bmatrix} \tag{3.36}$$

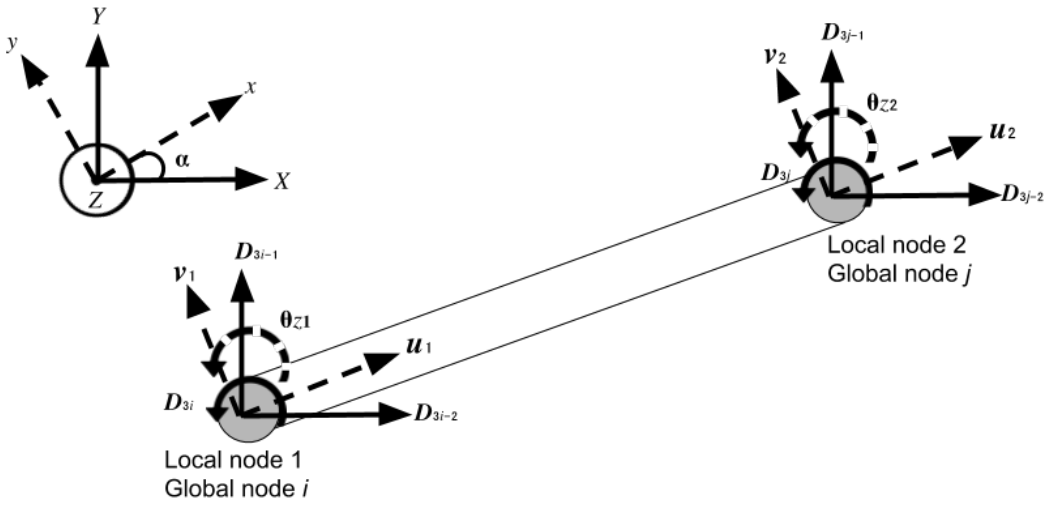


Figure 3.4: 2D Frame element expressed in a global coordinate system

The figure above shows how the local ($x-y$) coordinate system for a given element is related to the global ($X-Y$) coordinate system by a rotation α. This coordinate transformation relationship relating the local and global displacement vectors is defined by:

$$\boldsymbol{d_e} = \boldsymbol{T}\boldsymbol{D_e} \tag{3.37}$$

where $\boldsymbol{T}$ is a transformation matrix given by

$$\boldsymbol{T} = \begin{bmatrix} l_x & m_x & 0 & 0 & 0 & 0 \\ l_y & m_y & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & l_x & m_x & 0 \\ 0 & 0 & 0 & l_y & m_y & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

In 3.38 above, l_x, l_y, m_x, and m_y are defined by the coordinate relationship as:

$$\begin{aligned} l_x &= \cos(x, X) = \cos(\alpha) = \frac{X_j - X_i}{L} \\ l_y &= \cos(y, X) = \cos(90^\circ + \alpha) = -\sin(\alpha) = -\frac{Y_j - Y_i}{L} \\ m_x &= \cos(x, Y) = \sin(\alpha) = \frac{Y_j - Y_i}{L} \\ m_y &= \cos(y, Y) = \cos(\alpha) = \frac{X_j - X_i}{L} \end{aligned} \tag{3.39}$$

The length of the element is calculated as:

$$L = \sqrt{(X_j - X_i)^2 + (Y_j - Y_i)^2} \tag{3.40}$$

By utilizing the transformation matrix, the element stiffness matrix ($\boldsymbol{K_e}$) and force vector ($\boldsymbol{F_e}$) in the global coordinate system can be related to the local coordinate system by:

$$\boldsymbol{K_e} = \boldsymbol{T^T k_e T} \tag{3.41}$$

$$\boldsymbol{F_e} = \boldsymbol{T^T f_e} \tag{3.42}$$

3.3.2 3D Frame Element

So far in this chapter, bar, beam and frame elements have all been discussed within a 2D environment. While this serves a very important purpose for analytical approaches, it is not practical for real world applications which require 3D implementations. For a 3D frame element of length L, each node has 6 degrees of freedom (3 translational (u, v, w) and 3 rotational ($\theta_x, \theta_y, and \theta_z$) for a total of 12 DOF for the entire element. Figure 3.5 schematically shows this 3D concept. It is worth noting here that while the frame element may initially exist as a long straight slender member (see the cylinder in Figure 3.5), its deformed configuration will generally be represented by a cubic function in 3D space. Visualizing these deformed elements as straight cylinders is sufficient to represent the deformation of the structure with respect to nodal motions, but it does not accurately depict the deformed shape of the elements connecting these nodes. The element

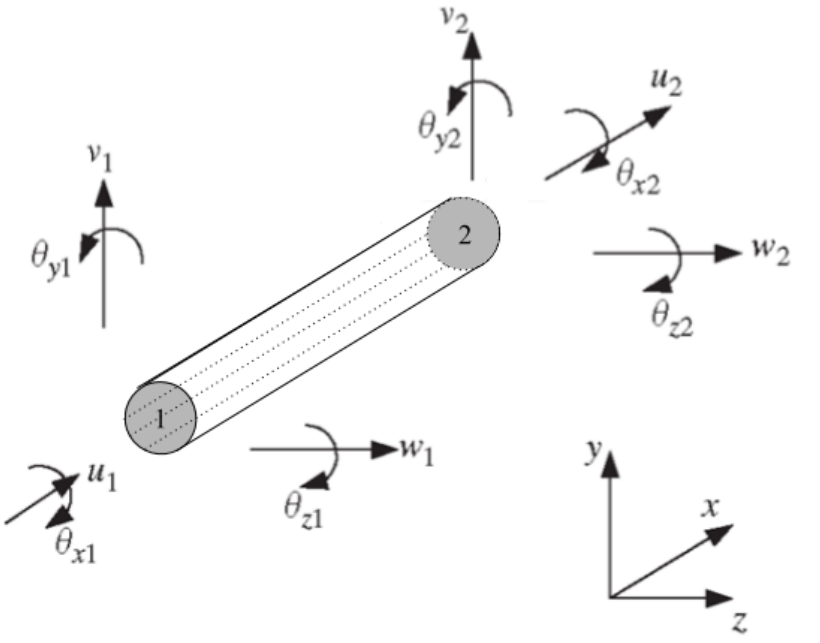


Figure 3.5: Example of a 3D frame element with six degrees of freedom per node (three translational, three rotational)

displacement vector ($\boldsymbol{d_e}$) for this element can be written as

$$\boldsymbol{d_e} = \left\{ \begin{matrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \\ d_{10} \\ d_{11} \\ d_{12} \end{matrix} \right\} = \left\{ \begin{matrix} u_1 \\ v_1 \\ w_1 \\ \theta_{x_1} \\ \theta_{y_1} \\ \theta_{z_1} \\ u_2 \\ v_2 \\ w_2 \\ \theta_{x_2} \\ \theta_{y_2} \\ \theta_{z_2} \end{matrix} \right\} \tag{3.43}$$

The element stiffness ($\boldsymbol{k_e}$) matrix can be obtained by adding together the 3D bar element matrix and 3D beam element matrix to get the matrix in Equation 3.44.

$$
\boldsymbol{k_e} = \begin{array}{c}
\begin{matrix} u_1 & v_1 & w_1 & \theta_{x_1} & \theta_{y_1} & \theta_{z_1} & u_2 & v_2 & w_2 & \theta_{x_2} & \theta_{y_2} & \theta_{z_2} \end{matrix} \\
\begin{bmatrix}
\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{-EA}{L} & 0 & 0 & 0 & 0 & 0 \\
 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\
 & & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 \\
 & & & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 \\
 & & & & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\
 & & & & & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\
 & & & & & & \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & \frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\
 & & & & & & & & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\
 & & symm. & & & & & & & \frac{GJ}{L} & 0 & 0 \\
 & & & & & & & & & & \frac{4EI_y}{L} & 0 \\
 & & & & & & & & & & & \frac{4EI_z}{L}
\end{bmatrix}
\end{array}
\tag{3.44}
$$

A couple key differences from the 2-D frame element are the usage of the second moment of inertia for both the z and y axis, (I_z and I_y), as well as cross-sectional polar moment of inertia (J) . For an element with a circular cross section, the I_z, I_y and J are:

$$
\begin{aligned}
I_z = I_y &= \frac{\pi r^4}{4} \\
J &= \frac{\pi r^4}{2}
\end{aligned}
\tag{3.45}
$$

Also worth noting that the 4th and 10th degrees of freedom (θ_{x1}, θ_{x2}) for the 3-D Frame element relate to the torsional deformation. The formulation of torsion is very similar to the formulation for axial deformation, with the replacement of the element tensile stiffness ($\frac{EA}{L}$) with the element torsional stiffness ($\frac{GJ}{L}$) where G is the shear modulus.

As seen in the section 2.3.1, some transformations will be needed to make a local element fit within a global coordinate system. These equations will be similar to the ones defined in Equations 3.37 - 3.42, with modifications to account for the 3 dimensionality of this transformation. This will be done through a 12×12 Transformation Matrix ($\boldsymbol{T}$)

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{\lambda} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\lambda} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\lambda} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{\lambda} \end{bmatrix} \tag{3.46}$$

consisting of multiple 3×3 $\boldsymbol{\lambda}$ matrices:

$$\boldsymbol{\lambda} = \begin{bmatrix} \cos(x,X) & \cos(x,Y) & \cos(x,Z) \\ \cos(y,X) & \cos(y,Y) & \cos(y,Z) \\ \cos(z,X) & \cos(z,Y) & \cos(z,Z) \end{bmatrix} \tag{3.47}$$

The terms inside $\boldsymbol{\lambda}$ represent the coordinate relationship between the local $x-y-z$ system and the global $X-Y-Z$ system. Note that the $\mathbf{0}$ entries represents 3×3 matrices consisting of only zeros. Equation 3.41 from above can now be applied to obtain the element stiffness matrix ($\boldsymbol{K_e}$) and the element force vector ($\boldsymbol{F_e}$) for this element in the global coordinate system as

$$\boldsymbol{K_e} = \boldsymbol{T^T k_e T} \tag{3.48}$$

$$\boldsymbol{F_e} = \boldsymbol{T^T f_e} \tag{3.49}$$

The stress at each node within the element and in turn the stress in the element can be found as

follows:

$$\begin{aligned}\sigma_{topN1} &= \left| \frac{F_{u1}}{A} - \frac{F_{\theta_{z_1}} r}{I_z} \right| \\ \sigma_{topN2} &= \left| \frac{F_{u2}}{A} - \frac{F_{\theta_{z_2}} r}{I_z} \right| \\ \sigma_{top} &= \max(\sigma_{topN1}, \sigma_{topN2})\end{aligned} \tag{3.50}$$

$$\begin{aligned}\sigma_{bottomN1} &= \left| \frac{F_{u1}}{A} + \frac{F_{\theta_{z_1}} r}{I_z} \right| \\ \sigma_{bottomN2} &= \left| \frac{F_{u2}}{A} + \frac{F_{\theta_{z_2}} r}{I_z} \right| \\ \sigma_{bottom} &= \max(\sigma_{bottomN1}, \sigma_{bottomN2})\end{aligned} \tag{3.51}$$

$$\begin{aligned}\sigma_{leftN1} &= \left| \frac{F_{v1}}{A} - \frac{F_{\theta_{z_1}} r}{I_z} \right| \\ \sigma_{leftN2} &= \left| \frac{F_{v2}}{A} - \frac{F_{\theta_{z_2}} r}{I_z} \right| \\ \sigma_{left} &= \max(\sigma_{leftN1}, \sigma_{leftN2})\end{aligned} \tag{3.52}$$

$$\begin{aligned}\sigma_{rightN1} &= \left| \frac{F_{v1}}{A} + \frac{F_{\theta_{z_1}} r}{I_z} \right| \\ \sigma_{rightN2} &= \left| \frac{F_{v2}}{A} + \frac{F_{\theta_{z_2}} r}{I_z} \right| \\ \sigma_{left} &= \max(\sigma_{rightN1}, \sigma_{rightN2})\end{aligned} \tag{3.53}$$

$$\sigma_{max} = \max(\sigma_{top}, \sigma_{bottom}, \sigma_{left}, \sigma_{right}) \tag{3.54}$$

This section discussed the finite element formulation of various element types and how to set up a simple system consisting of 2 nodes and 1 element. However, this is not a very useful solver, as most structures are more complex and composed of multiple elements and this section so far has not covered how multiple elements are assembled into the global structural context. Therefore to conduct finite element analysis of a structure with multiple elements, an iterative approach will need to be implemented and detailed in the section below.

3.4 Generic FEA Structural Solver Implementation

For the solving of a generic multi element structure, n (number of nodes) and m (number of elements between the nodes) within the structure will first need to be quantified. For this implementation, 3D frame elements will be used which means that every node has 6 degrees of freedom. A new variable g_{dof} will be implemented to keep track of the total degrees of freedom found within the structure and is found by:

$$g_{dof} = 6 \times n \tag{3.55}$$

A new global stiffness matrix ($\boldsymbol{K_g}$) as well as force ($\boldsymbol{F_g}$) and displacement vectors ($\boldsymbol{D_g}$) can be formulated as shown below. Also worth noting is that the counting syntax is defined to start at 0 as the first index. This is done as a convenience to the algorithmic approach to solving FEA using a popular programming language such as python.

$$\boldsymbol{K_g} = \begin{bmatrix} K_{0,0} & \dots & K_{0,g_{dof}-1} \\ \vdots & \ddots & \vdots \\ K_{g_{dof}-1,0} & \dots & K_{g_{dof}-1,g_{dof}-1} \end{bmatrix} \tag{3.56}$$

$$\boldsymbol{F_g} = \begin{Bmatrix} F_0 \\ \vdots \\ F_{g_{dof}-1} \end{Bmatrix} \tag{3.57}$$

$$\boldsymbol{D_g} = \begin{Bmatrix} D_0 \\ \vdots \\ D_{g_{dof}-1} \end{Bmatrix} \tag{3.58}$$

These matrices and vectors above are empty at the moment but will systematically be filled in with information from the individual elements that exist across various nodes. However, some material properties for the cylindrical element such as Young's modulus (E), Poisson's ratio (ν) and uniform radius (r) need to be defined first. Some basic properties can now be solved using the

equations from the previous chapter as follows:

$$\begin{aligned} A &= \pi r^2 \\ I_z = I_y &= \frac{\pi r^4}{4} \\ J &= \frac{\pi r^4}{2} \\ G &= \frac{E}{2(1+\nu)} \end{aligned} \tag{3.59}$$

For each node i that exists in the range from 0 to $n-1$, the solver must go through and iteratively pass the nodal forces to the global force vector using the following relationship:

$$\boldsymbol{f_i} = \begin{Bmatrix} f_{u_i} \\ f_{v_i} \\ f_{w_i} \\ f_{\theta_{x_i}} \\ f_{\theta_{y_i}} \\ f_{\theta_{z_i}} \end{Bmatrix} = \begin{Bmatrix} F_{6i} \\ F_{6i+1} \\ F_{6i+2} \\ F_{6i+3} \\ F_{6i+4} \\ F_{6i+5} \end{Bmatrix} \tag{3.60}$$

It is also important to specify nodal boundary conditions ($\boldsymbol{BC_i}$) which denote whether or not the node is fixed or free to translate and rotate in the $X-Y-Z$ axes. It is now necessary to make a global vector to keep track of this and the nodal conditions will be utilized later in this section.

$$\boldsymbol{BC_g} = \begin{Bmatrix} \boldsymbol{BC_0} \\ \vdots \\ \boldsymbol{BC_{n-1}} \end{Bmatrix} \; where \; \boldsymbol{BC_i} = \begin{Bmatrix} BC_{u_i} \\ BC_{v_i} \\ BC_{w_i} \\ BC_{\theta_{x_i}} \\ BC_{\theta_{y_i}} \\ BC_{\theta_{z_i}} \end{Bmatrix} = \begin{Bmatrix} BC_{6i} \\ BC_{6i+1} \\ BC_{6i+2} \\ BC_{6i+3} \\ BC_{6i+4} \\ BC_{6i+5} \end{Bmatrix} \tag{3.61}$$

Algorithm 1 shows a pseudo-code for filling of the global force and boundary condition vectors.

Algorithm 1 Initialize generic solver and apply nodal properties

Require: `n > 0 && m >0`

$g_{dof} = 6n$

K_g `= new Matrix` $\left[g_{dof} \times g_{dof}\right]$

F_g `= new Vector` $\left\{g_{dof} \times 1\right\}$

D_g `= new Vector` $\left\{g_{dof} \times 1\right\}$

BC_g `= new Vector` $\left\{g_{dof} \times 1\right\}$

$A = \pi r^2$

$I_z = I_y = \frac{\pi r^4}{4}$

$J = \frac{\pi r^4}{2}$

$G = \frac{E}{2(1+\nu)}$

for `i = 0; i <` n`; i++` **do**

  for `a = 0; a < 6; a++` **do**

    $F_g[6i + a] = f_i[a]$

    $BC_g[6i + a] = BC_i[a]$

  end for

end for

Now the solver will need to iteratively create element stiffness matrices (k_e) for each element j that we have in the range from 0 to $m - 1$. Each element j exists between two nodes A and B. A new elemental degree of freedom vector ($\boldsymbol{DOF_j}$) will be created that keeps track of the 6 degrees

of freedom from node A and the 6 degrees of freedom from node B.

$$\boldsymbol{DOF_j} = \begin{Bmatrix} DOF_{6A} \\ DOF_{6A+1} \\ DOF_{6A+2} \\ DOF_{6A+3} \\ DOF_{6A+4} \\ DOF_{6A+5} \\ DOF_{6B} \\ DOF_{6B+1} \\ DOF_{6B+2} \\ DOF_{6B+3} \\ DOF_{6B+4} \\ DOF_{6B+5} \end{Bmatrix} \tag{3.62}$$

The length of the element (L) can be calculated using the $X - Y - Z$ position of the nodes A and B in the global coordinate system.

$$L = \sqrt{(X_B - X_A)^2 + (Y_B - Y_A)^2 + (Z_B - Z_A)^2} \tag{3.63}$$

The $\boldsymbol{k_j}$ stiffness matrix for this element in the local coordinate system can now be solved as shown

in 3.44 above.

$$
\boldsymbol{k_j} = \begin{array}{c}
\begin{array}{cccccccccccc}
u_A & v_A & w_A & \theta_{x_A} & \theta_{y_A} & \theta_{z_A} & u_B & v_B & w_B & \theta_{x_B} & \theta_{y_B} & \theta_{z_B}
\end{array} \\
\left[\begin{array}{cccccccccccc}
\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{-EA}{L} & 0 & 0 & 0 & 0 & 0 \\
 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\
 & & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 \\
 & & & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 \\
 & & & & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\
 & & & & & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\
 & & & & & & \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & \frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\
 & & & & & & & & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\
 & & symm. & & & & & & & \frac{GJ}{L} & 0 & 0 \\
 & & & & & & & & & & \frac{4EI_y}{L} & 0 \\
 & & & & & & & & & & & \frac{4EI_z}{L}
\end{array}\right]
\end{array}
\tag{3.64}
$$

This matrix will then need to be transformed from the local coordinate system into the global coordinate system and is done through the use of Equation 3.47 and 3.46 as explained in the Section 3.3. However, since the elements exist in a 3-D environment, a couple situations appear that affect how $\boldsymbol{\lambda}$ is formulated. If left unresolved, these corner cases might return a value of 0 that could lead certain programming languages to crash. These are the three primary paths to the programmatic formulation of $\boldsymbol{\lambda}$. If $X_A == X_B$ and $Y_A == Y_B$, this means that the element exist in the same $X - Y$ plane and the only difference between the two nodes is a change of position in the Z axis.

1. If $Z_B > Z_A$:

$$
\boldsymbol{\lambda} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \tag{3.65}
$$

2. If $Z_B < Z_A$:

$$\boldsymbol{\lambda} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.66}$$

3. If $X_A == X_B \,\text{and}\, Y_A == Y_B \,\text{and}\, Z_A == Z_B$, then the nodes are in the exact same location and an element cannot be created there.

4. If the defined above cases do not arise, and $X_A \,!= X_B$ or $Y_A \,!= Y_B$, then $\boldsymbol{\lambda}$ can be solved using 3.47 from the previous section.

Algorithm 2 shows the psuedo-code for the generation of the global stiffness matrix.

Algorithm 2 Generically solve for element stiffness $\boldsymbol{K_j}$ for all elements in the global coordinate system

Require: `n > 0 && m >0`

for `j = 0; j < m; j++` **do**

k_j `= new Matrix` $\left[12 \times 12\right]$

f_j `= new Vector` $\left\{12 \times 1\right\}$

DOF_j `= new Vector` $\left\{12 \times 1\right\}$

`A =` $m.Node1$

`B =` $m.Node2$

for `a = 0; a < 6; a++` **do**

$DOF_j[a] = A.DOF[a]$

$DOF_j[6 + a] = B.DOF[a]$

end for

$L = \sqrt{(B_X - A_X)^2 + (B_Y - A_Y)^2 + (B_Z - A_Z)^2}$

`<Solve for` $\boldsymbol{k_j}$ `using Equation 3.64 above>`

if $A_x == B_x$ **&&** $A_y == B_y$ **then**

if $B_z > A_z$ **then**

`<Solve for` $\boldsymbol{\lambda}$ `using Equation 3.65>`

else

`<Solve for` $\boldsymbol{\lambda}$ `using Equation 3.66>`

end if

else

`<Solve for` $\boldsymbol{\lambda}$ `using Equation 3.47>`

end if

`<Solve for` $\boldsymbol{T}$ `using Equation 3.46>`

$\boldsymbol{K_j = T^T k_j T}$

$\boldsymbol{F_j = T^T f_j}$

`<Algorithm 3>`

end for

Now that each element j has an element stiffness matrix in the global coordinate system ($\boldsymbol{K_j}$), their individual stiffness matrices should be added into the global stiffness matrix of the structure ($\boldsymbol{K_g}$). This will be done through the use of the $\boldsymbol{DOF_j}$ vector that was defined earlier. $\boldsymbol{K_j}$ is a 12×12 matrix and the $\boldsymbol{DOF_j}$ is a 12×1 vector. The global $\boldsymbol{K_g}$ matrix is sized as being g_{dof} by g_{dof}. The key component here is to then match up the individual nodes of element j and add the component from $\boldsymbol{K_j}$ into $\boldsymbol{K_g}$. This is usually done through a nested for-loop and is briefly shown in pseudo-code of Algorithm 3. Once Algorithm 2 is completed, the boundary

Algorithm 3 Assembling element stiffness $\boldsymbol{K_j}$ into global stiffness $\boldsymbol{K_g}$

```
for a = 0; a < 12; a++ do
  for b = 0; b < 12; b++ do
    K_g[DOF_j[a], DOF_j[b]] = K_g[DOF_j[a], DOF_j[b]] + K_j[a, b]
  end for
end for
```

condition vector for each node ($\boldsymbol{BC_i}$) is applied. If any component of the nodal $\boldsymbol{BC_i}$ vector $\{BC_{u_i}, BC_{v_i}, BC_{w_i}, BC_{\theta_{x_i}}, BC_{\theta_{y_i}}, BC_{\theta_{z_i}}\}^T$ is fixed, the rows and columns of the K_g matrix associated with that component will need to be zeroed out. However, in order to maintain non-singularity of the matrix, the intersection of that row and column (value along the diagonal) will be replaced with a 1 instead of a zero. Also if a component of $\boldsymbol{BC_i}$ is fixed, there can be no force applied in that component, thus the corresponding row in $\boldsymbol{F_g}$ is zeroed out. The application of the boundary conditions to the element matrix and force vectors is summarized in Algorithm 4.

Algorithm 4 Applying Boundary Conditions

$\hat{\boldsymbol{K}_g} = \boldsymbol{K_g}$

$\hat{\boldsymbol{F}_g} = \boldsymbol{F_g}$

for `i = 0; i < n; i++` **do**

  for `b = 0; b < 6; b++` **do**

    if $\boldsymbol{BC_i}[b] == FIXED$ **then**

      for `c = 0; c <` g_{dof}`; c++` **do**

        $\hat{\boldsymbol{K}_g}[c,\ 6i + b] = 0$

        $\hat{\boldsymbol{K}_g}[6i + b,\ c] = 0$

      end for

      $\hat{\boldsymbol{K}_g}[6i + b,\ 6i + b] = 1$

      $\hat{\boldsymbol{F}_g}[6i + b] = 0$

    end if

  end for

end for

`Solve for displacement(`$\boldsymbol{D_g}$`) via:` $\hat{\boldsymbol{F}_g} = \hat{\boldsymbol{K}_g}\boldsymbol{D_g}$

`Compute reactionary forces (`$\boldsymbol{F_g}$`) via:` $\boldsymbol{F_g} = \boldsymbol{K_g}\boldsymbol{D_g}$

At this point, the complete stiffness of the structure is formulated and global displacement vector ($\boldsymbol{D_g}$)can be obtained by solving the following:

$$\boldsymbol{F_g} = \boldsymbol{K_g}\boldsymbol{D_g} \tag{3.67}$$

Keep in mind that inverting matrices are not the fastest or most optimized way of solving equations, and the implementation of a LU Decomposition or Cholesky Solver would greatly improve the computation time [63, 64]. The stresses for each element is found using equations 3.50 to 3.54. This process is summarized in Algorithm 5.

Algorithm 5 Finding maximum stress within elements

for `i = 0; i < m; i++` **do**

`A =` $m.Node1$

`B =` $m.Node2$

$\sigma_{topA} = \left| \frac{F_g[6A]}{A_e} - \frac{F_g[6A+5]r}{I_z} \right|$

$\sigma_{topB} = \left| \frac{F_g[6B]}{A_e} - \frac{F_g[6B+5]r}{I_z} \right|$

$\sigma_{top} = \max(\sigma_{topA}, \sigma_{topB})$

$\sigma_{bottomA} = \left| \frac{F_g[6A]}{A_e} + \frac{F_g[6A+5]r}{I_z} \right|$

$\sigma_{bottomB} = \left| \frac{F_g[6B]}{A_e} + \frac{F_g[6B+5]r}{I_z} \right|$

$\sigma_{bottom} = \max(\sigma_{bottomA}, \sigma_{bottomB})$

$\sigma_{leftA} = \left| \frac{F_g[6A+1]}{A_e} - \frac{F_g[6A+5]r}{I_z} \right|$

$\sigma_{leftB} = \left| \frac{F_g[6B+1]}{A_e} - \frac{F_g[6B+5]r}{I_z} \right|$

$\sigma_{left} = \max(\sigma_{leftA}, \sigma_{leftB})$

$\sigma_{rightA} = \left| \frac{F_g[6A+1]}{A_e} + \frac{F_g[6A+5]r}{I_z} \right|$

$\sigma_{rightB} = \left| \frac{F_g[6B+1]}{A_e} + \frac{F_g[6B+5]r}{I_z} \right|$

$\sigma_{left} = \max(\sigma_{rightA}, \sigma_{rightB})$

$\sigma_{max} = \max(\sigma_{top}, \sigma_{bottom}, \sigma_{left}, \sigma_{right})$

end for

A complete approach to how these algorithms work together to make a generic FEA solver is shown below. For a detailed look at the implementation of this in $C\#$, look at the code attached in Appendix A. The next chapter will discuss how this solver will be implemented in Virtual Reality environments, and the user interaction tools that will be designed.

Algorithm 6 Complete algorithmic approach to finite element analysis

```
Pre-process structure:  Algorithm 1

Analyze structure:  Algorithms 2,3 and 4

Post-process structure:  Algorithm 5
```

4. VIRTUAL REALITY IMPLEMENTATION OF FINITE ELEMENT ANALYSIS

The previous chapter explained how finite element analysis of 3D frame elements is conducted. This chapter will explain how an FEA solver can be implemented inside virtual reality environments for realtime structural analysis capabilities. The A-Frame and Unity environments will be explored for this work. Both environments provide significant benefits and advantages that the other environment does not provide by default.

A-Frame is an emerging web framework that enables the development and embedding of virtual reality experiences directly within a website. Modern day web pages are written with *HTML* handling the user experience front-end and with *JavaScript* handling all the back-end of the data input/output by the user. A-Frame exists as an easy to implement framework within the typical web development workflow that handles all the VR setup, controls and interactions. This process requires no install or build steps by the user or the developer. A-Frame is thus enabled to truly be cross-platform as any device that has a web browser can access and interact with the website and A-Frame takes care of all the virtual reality specifics. Development with the A-Frame framework is also streamlined as pre-fabricated geometries can be imported as FBX objects within the HTML front-end and visualized immediately. Initially created as an in-house tool at Mozilla, A-Frame was later made open-source and available to the community. This technology has been implemented by many companies, such as Google, Toyota, NASA, etc. and is continuously being improved. A-Frame simulations are beneficial for teaching purposes as VR headsets are not extremely common at the moment and reduces the barrier to entry as not all students have one. The disadvantages come from the lack of software development kits that would enable higher user interaction and capabilities as well as the computational limits of the simulation being capped by the capabilities of the device using it.

Unity is a game development engine that can be used to make 2D, 3D, VR/AR games and simulations. This engine has been around for 15+ years and is capable of developing for many platforms. Unity utilizes $C\#$ as its main programming language, and has a very strong developer

network. The reason why this engine was chosen over its competitors like Unreal Engine and Autodesk Stingray is because of the many open source software development kits and community troubleshooting available for Unity. This has made Unity a standard for many industry professionals. The advantages are that its easy of use for simulation development as well as its built in optimization tools. the downsides are that access to it is restricted to only users with a VR capable headset as well as a powerful computer.

In this chapter, workflows for how the user and virtual reality simulation interact with the FEA solver in each environment will be explained. User interactions will be designed to enable structural manipulation and creation capabilities. Research will be done to improve the efficiency of the solver in order to reduce the latency within the virtual reality environments. To test the accuracy of the solvers implemented within this work, their results will be compared to that of ABAQUS, a commercial FEA software. Insight gleaned from the various preliminary VR development efforts of Chapter 2 are itemized below and were considered throughout the simulation design process as they provide valuable input in making simulations more immersive and user friendly.

- Development of an effective but diverse experiences will require a well documented SDK and a strong development community.

- User interaction is valuable and users both enjoy and benefit from the ability to use their hands/controllers to manipulate objects.

- Users were interested in the capability to visualize stress contours; it is important that a color scheme for stress that is easily distinguishable from a glance be implemented in this work.

- Users communicated the need for an on-screen user interface as the single button on the Google Cardboard device does not allow for sufficient user interaction capabilities. Gaze interaction and single-button selection of items from a menu, however, could greatly increase user interaction options.

- When the model to be visualized was complex/large, longer load times were required on mobile devices and the rendering of even simple motions could be performed only with very

high latency. Further investigation showed that A-Frame uses the processor of the device to parse the web-page and perform all graphical operations so most handheld devices do not have near the requisite computing power. Therefore, models rendered in such environments would need to be relatively small.

4.1 A-Frame Environment

To utilize the A-Frame environment to build virtual reality experiences for the web, some tools are needed. The tools used for this thesis are:

- Python v3.7- This will be utilized to install necessary development packages and host a web server to test the website locally.
- Atom IDE - Any software integrated development environment will work, however, Atom has some useful GitHub control and debugging tools.
- Mozilla Firefox - Although A-Frame runs on all web browsers, Mozilla created the framework and has optimized their browser for webVR experiences.
- GitHub Pages or Glitch - For any person to visit the webpage from their respective devices, the website will first need to be hosted online. This can be a confusing part of this process but Glitch.com and GitHub Pages have free and easy to use web hosting services. If there are many concurrent users, these services may not be the best performing, but will be able to at minimum meet the performance needs of this work.
- VR Head-mounted Display (HMD) - While this is not necessary to use WebVR, these devices provide a more immersive VR experience. There are many HMD's that could ahve been utilized but the work done in this thesis was tested to work on both the HTC Vive and the Oculus Quest.

Within the *HTML / JavaScript* development environment, certain open-source packages were utilized within the work of this thesis. These packages are:

- *aframe.js* - This is the overall A-Frame package that lets the browser know that a WebVR experience could be implemented within the website. It contains all the prerequisite VR controls and handling that serve as the building blocks for future scripts.

- *aframe-extras.js* - This package contains add-ons and helper scripts that enable functionality such as model loading and movement for non HMD users.

- *aframe-physics-system.js* - This package enables the physics within the VR environment and allows capabilities such as dynamic and static objects.

- *aframe-teleport.js* - This package allows the users wearing HMDs to teleport around the scene.

- *aframe-input-mapping.js* - This package allows multiple devices to all register actions. A input is mapped across all buttons of defined types (HMD controller trigger, mouse click, touchscreen press) to all conduct the same/different actions in the experience as needed.

- *aframe-super-hands.js* - This package allows user to commit gestures such as hover, grab and collide. This is useful when designing user interactions in VR experiences.

- *dat-gui-vr.js* - This package implements a flexible user interface within A-Frame that lets users across platforms interact with objects.

- *three.js* - The underlying 3D visualization library that A-Frame is built upon. This library handles the creation of primitive objects, geometry and shaders.

- *math.js* - An extensive math library with many functions, as well as linear algebra capabilities built in.

The FEA Solver from the previous section was written in *JavaScript* and the visualizations implemented within HTML which is then served to the user. The workflow for this is shown below.

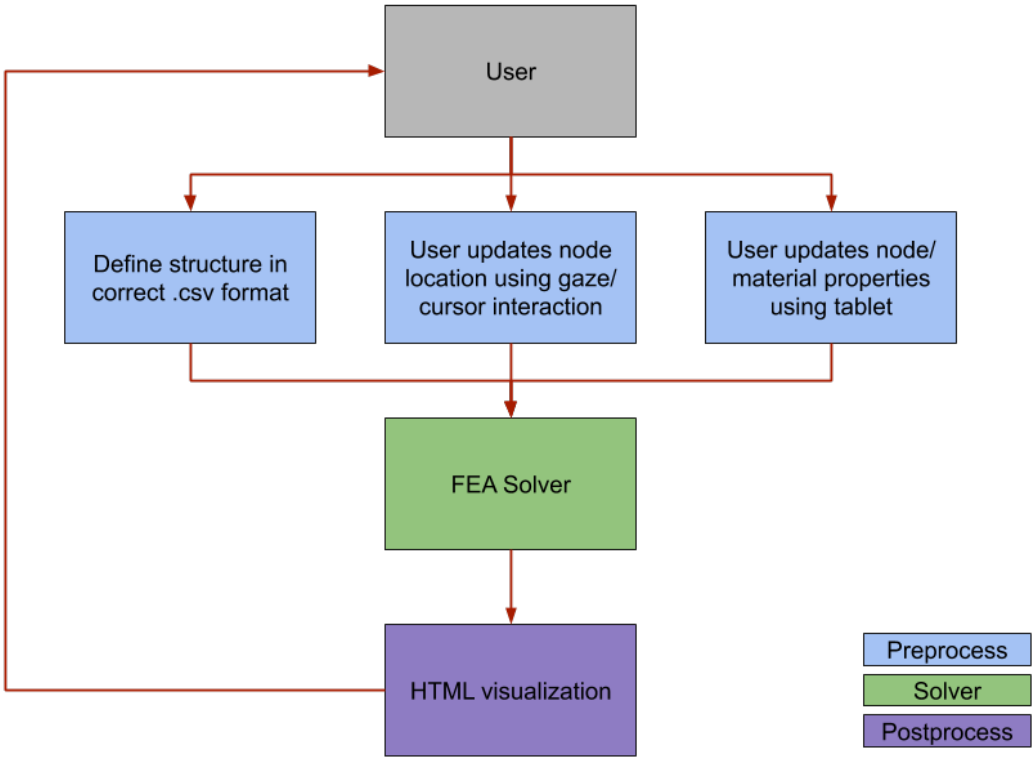


Figure 4.1: A-Frame user workflow

The subsections below will be broken into Preprocess, Solver, and Postprocess to further explain the pieces of 4.1. Preprocess will explain how nodes and elements are initialized into the environment as well as how users interact with them. Solver will cover how the system of elements and nodes are iteratively processed and solved using the work from Chapter 2. Postprocess will cover how the elements are visualized, and how user interaction is handled. That subsection will also delve into investigations made with multi-user networked environments as well as with solver accuracy and simulation latency.

4.1.1 Pre-Process

Structure Definition Through File IO

In order to conduct structural analysis, the nodes and elements of that structure have to first be defined. *nodeJS* is a server side *JavaScript* runtime environment. When a user loads up a website,

they initiate a call to the server hosting the site and the server processes and updates various pieces before serving the complete website back to the user. If the website takes too long to load, it usually means that there is a bottleneck on the server-side of the website that prevents it from serving users efficiently. For the work done in this thesis, the user would need to define the structure within a .csv file and also give that filename and reference to the *nodeJS* script that is in charge of parsing the file and serving the website. The beauty of .csv files are that it can be edited using a simple text editor like Notepad or with a more robust software such as Excel and still maintain its structure.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	*Node	xPos	yPos	zPos	xDOF	yDOF	zDOF	xRot	yRot	zRot	forceX [N]	forceY [N]	forceZ [N]
2	0	0.4	0.4	0.4	1	1	1	1	1	1	0	0	0
3	1	1.4	0.4	0.4	0	0	0	0	0	0	0	0	0
4	2	1.4	1.4	0.4	0	0	0	0	0	0	0	-0.5	0
5	3	0.4	1.4	0.4	1	1	1	1	1	1	0	0	0
6	4	0.4	0.4	0.9	1	1	1	1	1	1	0	0	0
7	5	1.4	0.4	0.9	0	0	0	0	0	0	0	0	0
8	6	1.4	1.4	0.9	0	0	0	0	0	0	0	0	0
9	7	0.4	1.4	0.9	1	1	1	1	1	1	0	0	0
10	8	0.4	0.4	1.4	1	1	1	1	1	1	0	0	0
11	9	1.4	0.4	1.4	0	0	0	0	0	0	0	0	0
12	10	1.4	1.4	1.4	0	0	0	0	0	0	0	0	0
13	11	0.4	1.4	1.4	1	1	1	1	1	1	0	0	0
14													
15	*Element	nodeA	nodeB										
16	0	0	1										
17	1	1	2										
18	2	2	3										
19	3	3	0										
20	4	4	5										
21	5	5	6										
22	6	6	7										
23	7	7	4										
24	8	0	4										
25	9	1	5										
26	10	2	6										
27	11	3	7										
28	12	0	8										

Figure 4.2: User definition of structure in .csv file

Within the image above, there are two distinct definition processes. *Node denotes to the *nodeJS* script that all entities below it are nodes. The *Element interrupts the *Node definition process and asserts that all entities below that are elements. Within the *Node entities, column A denotes the Node number, columns B through D denotes the $X - Y - Z$ positions, columns E through J are a binary system that denotes fixed(1) or free(0) for that degree of freedom, and columns K through M denote the amount of force at that node in Newtons. The *Element entities

are a bit simpler with column A denoting the element number and column B and C denoting the start and end node the element exists between.

The *nodeJS* script reads through all the data in the .csv files and adds them as entities within two separate JSON objects for nodes and elements. JSON (JavaScript Object Notation) is a simple and lightweight storage syntax that allows data to be easily passed between the server-side and the user-side. Once finished parsing, this data from the server is then sent to the FEA solver script and the webpage is served to the user.

Virtual Reality User Interface

To enable user manipulation of the defined structure within A-Frame, an open-source library called *dat-gui-vr.js* was utilized and is shown in the image below. This "tablet" style interface can be moved around by the user within the environment and can be used to modify the properties of the nodes and structure. The capabilities of the tablet interface are:

- Force sliders: These sliders can add forces in the $X - Y - Z$ direction for any of the nodes.
- DOF toggle: These toggles can manipulate the 6 degrees of freedom for each node and either set it free or fix it.
- Young's modulus: This drop down menu changes this property from any of the 6 pre-programmed choices(e.g. Polypropylene, Steel, Titanium, etc).
- Radius slider: This slider controls the radius of the elements and will update it in the solver as well as on the visualization.
- Maximum allowable stress: this component allows the changing of this property which changes the color scheme of the structure.
- Scale factor slider: This allows the manipulation of the structure from something small that the user can interact with easily to something large that the user can immerse themselves in.
- Deformed/undeformed toggle: This allows the user to visualize the structure structure as undeformed, deformed, or both simultaneously.

- Analysis button: This button send a command that triggers the Solver script to conduct the structural analysis.
- Reset button: this button allows the user to recall the original JSON objects from the server and reset the structure to its pre-defined setting.

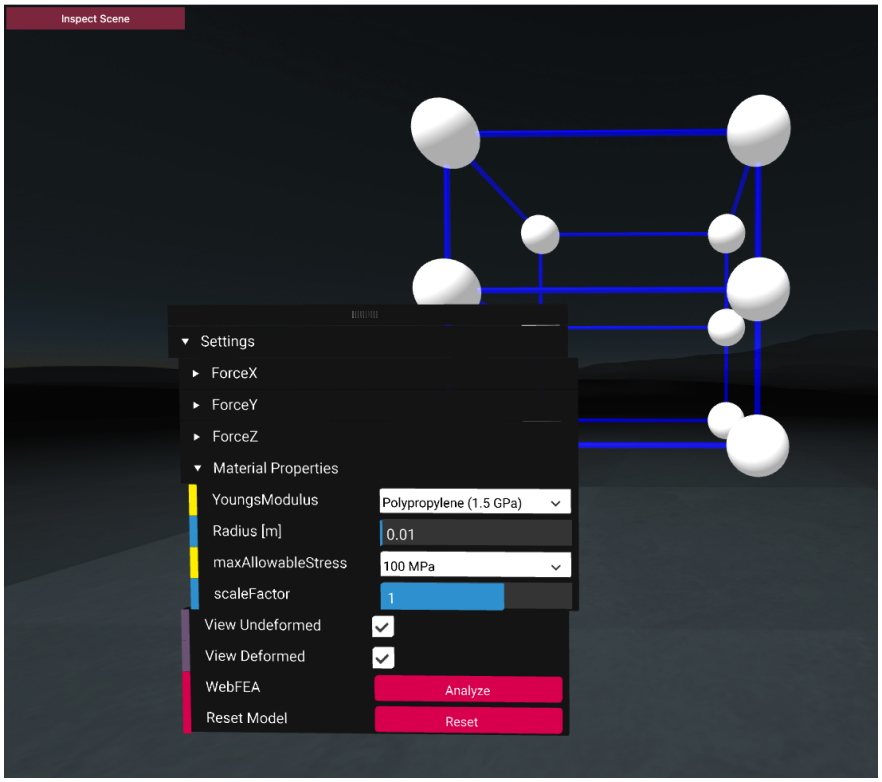
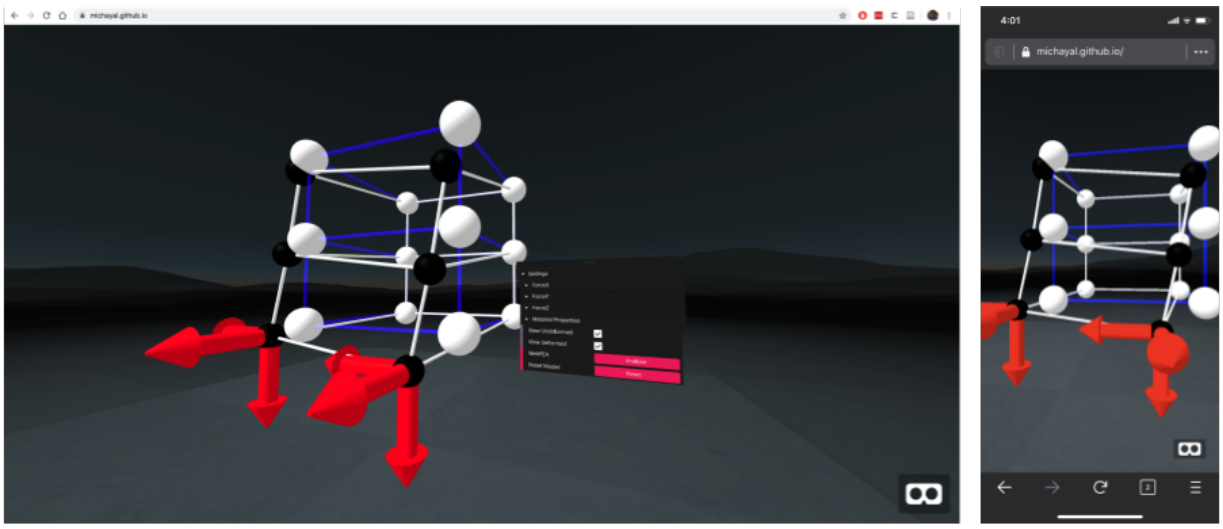


Figure 4.3: A-Frame user interaction tool

4.1.2 Solver

When information is passed to the FEA solver script, either from initial user definition, or from user interaction later on, it updates the internal JSON objects. Initially, structural analysis was conducted every time the information was updated but this was later changed to be user triggered. The reasoning for this was to reduce the computational overhead for when the user does an action such as dragging the slider to apply more force on a node. When called, the solver starts processing the information and solving the system of equations using the method described in 3.4. As this

has already been dutifully explained in the previous chapter, this subsection will only discuss the processes pertinent to the A-Frame implementation. Using an example defined similarly to Figure 4.2, this structure has 20 elements spread across 12 nodes as shown in the image below with the blue elements and the white nodes. This is a rather simple 3D structure and serves as a good benchmark test for the solver.



A-Frame on desktop browser

A-Frame on mobile browser

Figure 4.4: A-Frame structure in both mobile and desktop simulations

Once the JSON objects have been parsed and the $\boldsymbol{K_g}$ matrix has been found, displacement can be solved for using equation 3.67. However, in the *math.js* library, a LU decomposition linear algebra method exists. The traditional method of solving the system of equations by computing the matrix inverse of $\boldsymbol{K_g}$ took 44.55ms to complete. This is far too much time, as real-time VR requires that no process takes more than 20ms in order to prevent motion sickness and high latency. When the LU decomposition method was used, the solver was able to complete within 12.5ms. This is a much more usable speed, however, this structure is rather simple and increasing the structure by 10 elements and nodes caused the computation to go over the 20ms latency limit. While this is not an impact felt by desktop users, this is unusable for users on mobile platforms and those using a

HMD.

Once the solver does the analysis and finds where the structure deformed to after loads, it creates/updates two other JSON objects that hold the information for deformed nodes and elements. It then passes this information to the postprocessing to display in white elements and black nodes.

4.1.3 Post-Process

This section will explain the various components that go into the A-Frame simulation and the user interaction capabilities. Locomotion of the user within the environment is offloaded to be handled by A-Frame rather selected by the user. This is because the A-Frame system recognizes the user device and allows device dependent techniques. It will be able to provide WASD controls to desktop users, touch controls for mobile users, and teleportation controls to HMD users automatically. The entities shown within the analysis were also taken from the A-Frame library rather than brought in. These primitive objects are lightweight and easy to render and accurately able to depict the nodes and elements within the A-Frame environment. The nodes for the structure are represented by a similarly named A-Frame node entity. This entity is shown below and every node has the same built in programmable components.

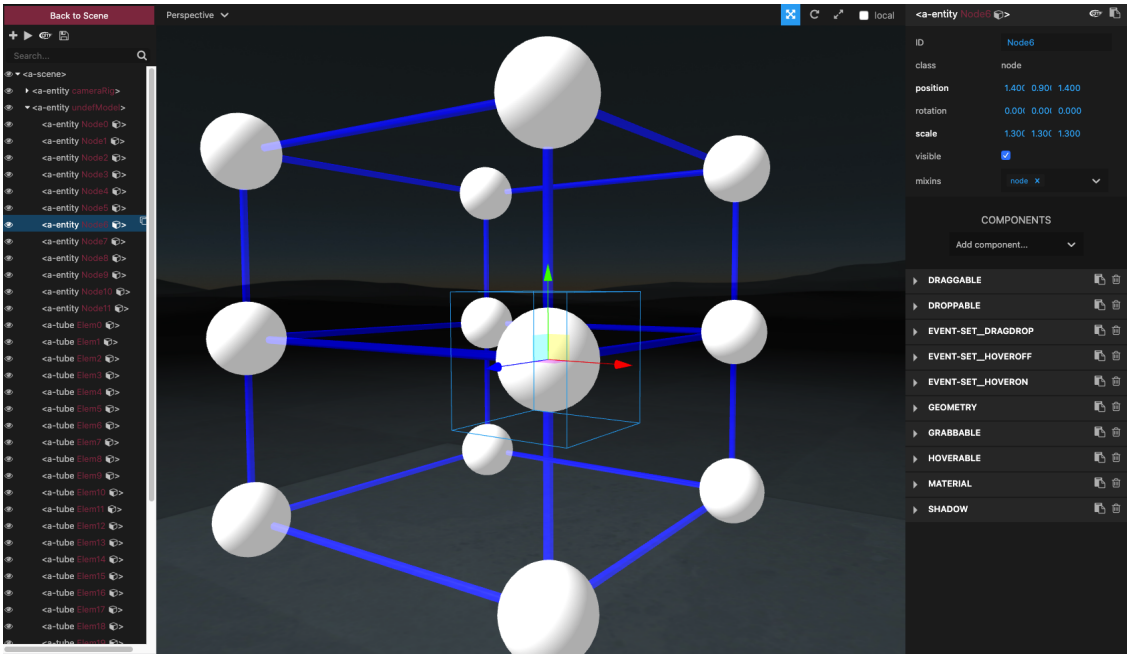


Figure 4.5: A-Frame node entity with components

These components emit a command when they are used which allows the programming of specific events. For example, when the node is drag-dropped to a new location, it is programmed to emit a command to update the JSON entry for that node within the Solver script. The various user interaction capabilities with the node are described below:

- Drag-drop: Allows users to grab a node and reposition it within the world.
- Hover: Allows user to receive pertinent knowledge such as node name or force magnitude at a glance.
- UI interaction: Allows the UI tool to apply forces and manipulate degrees of freedom for the node.

The elements are visualized using the A-Frame tube entity. The tube entity was chosen over the cylinder entity purely for programmatic instantiation purposes. The cylinder entity needed to be defined at a midpoint and the orientation had to be perfectly calculated every time. This proved unnecessarily cumbersome as the tube entity enabled the same components but with the ability to set a start point and an end point. The tube would then be created and oriented correctly between those points. Unlike the node entity however, all user interaction capabilities have been turned off for the undeformed element, as well as for the deformed element and tube. This is to prevent the user from committing an illegal action of moving an element without moving the nodes attached to it. While the above process accurately visualizes truss elements, frame elements are capable of bending and depicting them as long cylindrical tubes is an oversimplified visualization. Future work must consider the mathematical form of the element shape functions when determining and rendering their deformed geometric shapes.

The environment of the A-Frame simulation is kept simple to reduce the graphic workload for mobile users. There is a light placed directly above where the FEA structure will instantiate and the floor is gray in order to not distract from the structure. A 3D image is used to visualize a serene night sky for the backdrop of the environment. There is also a legend in the scene that shows the

different color gradients for stresses. This legend rotates to always face the user as shown in the image below.

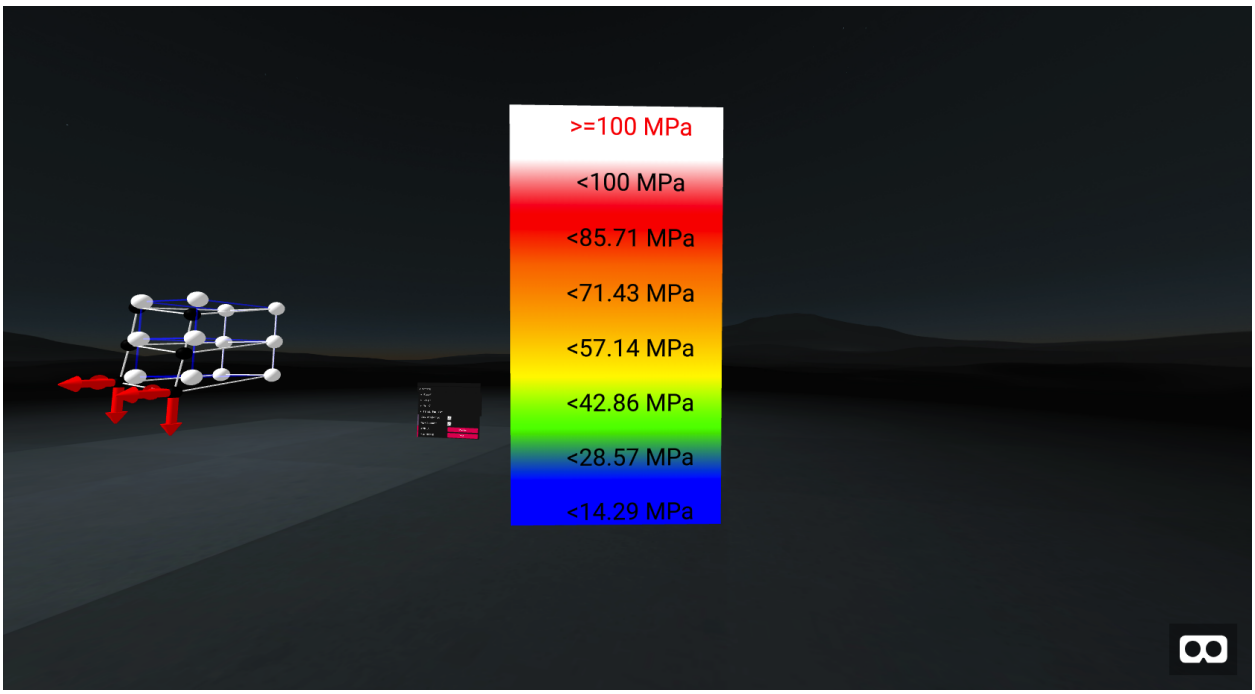


Figure 4.6: A simple A-Frame simulation environment

4.1.4 Further Investigations

The primary goal of conducting and visualizing structural analysis in A-Frame has been accomplished. However, it is important to further research the capabilities of A-Frame. To this end, multi-user interaction implementation was conducted. In earlier A-Frame projects, this had been rather easy to implement. Figure 3.7 is an image of multiple users all viewing a 3D Stress vs. Strain vs. Temperature plot of Shape Memory Alloys. The user avatars are rather simplistic but their gaze and movement are in perfect sync. However, when implemented within the project for this thesis, synchronous multi-user capabilities became much more complicated. In this work, all structural analysis has been done on the individual user devices. Therefore, when a user updates a node's position or applies a force, the calculations is all done internally. There is no communi-

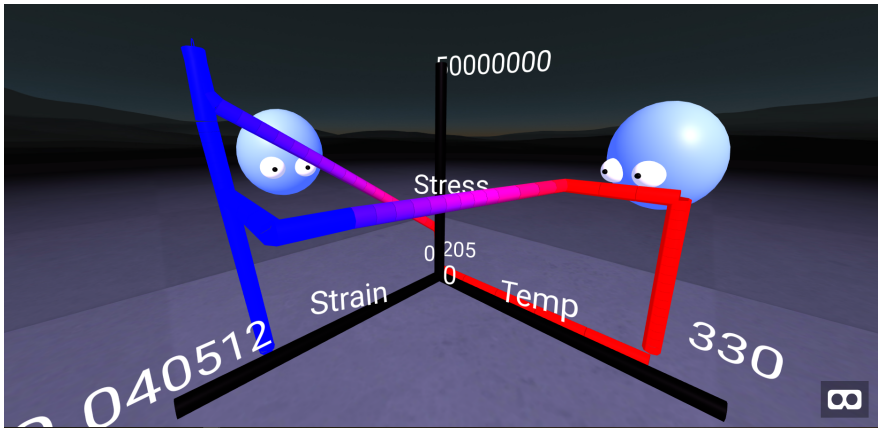


Figure 4.7: Multiple users in an A-Frame environment

cation between the user and the server regarding the structure. Although it is possible to have all the users hosted in the same environment for structural analysis, they cannot interact with the same structure. In order to implement that, a more complex server solution will be needed for hosting, and the structural analysis would have to be conducted there. The scope of this thesis does not delve into networked servers, so this is an area of improvement for future work.

An investigation was also conducted on the use of different element types within the solver. A new Solver script was written that would be capable of solving 2D Constant Strain Triangle (CST) elements. A plate with a hole comprised of these elements is visualized in the image below. While it is possible to use different element types, the latency for solving this structure was about 100ms which is outside the real-time constraint and would not be very user-friendly for a user wearing a HMD.

In conclusion, A-Frame is a very versatile environment for visualizing virtual reality environments on the web. However, it is not a very suitable tool for analyzing structures in real-time due to the latency involved with on-board computation. For this implementation to be more useful, the user interactions and analysis would have to be off-loaded to a server and the results would be brought back to be visualized within A-Frame.

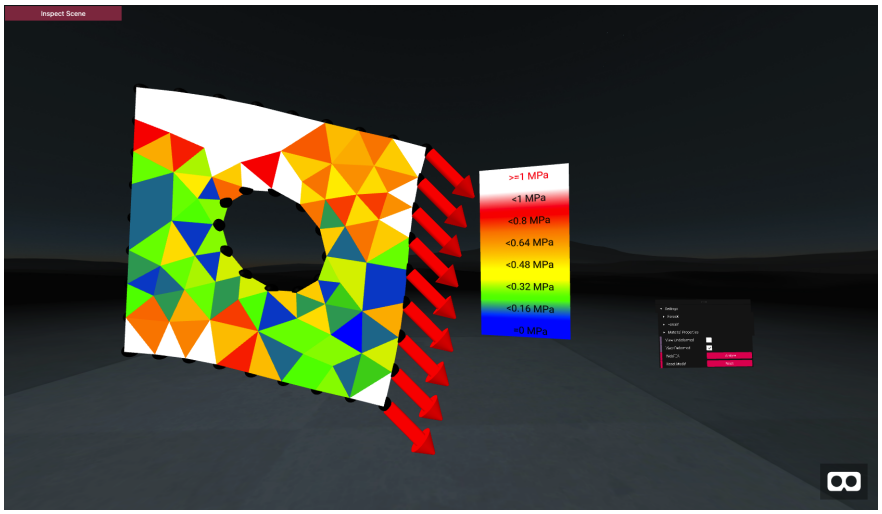


Figure 4.8: Analysis of a structure comprised of CST elements

4.2 Unity Environment

When developing for any specific device in Unity, that device's SDK must be imported and utilized. For Virtual Reality, the HTC Vive requires the SteamVR SDK, Oculus Rift requires the Oculus SDK, VR Simulator requires the Windows SDK, etc. This makes every application built within Unity device specific, and porting it over to another HMD would be complex and difficult. VRTK is an open source SDK utilized within this thesis that enables the work done to be easily compatible across devices. VRTK offers a common solution that enables developers to get started without all the effort of individual device setup. These scripts have embedded all three of the above mentioned SDKS into a common functionality codebase. Some of the capabilities enabled by the use of VRTK are:

- Device handling: VRTK automatically scans the connected devices and brings up the appropriate control system for the device used.

- Locomotion: Teleportation and traveling within virtual space have been simplified into button presses by the connected device.

- Interactions: Scripts can be added to objects to enable capabilities such as touching, grab-

bing, and using. This enables more complex interactions as one button can be pushed to grab an object, while a different button can be used to activate that object's functionality.

- UI elements: This allows a more robust handling of user interaction with UI elements. The same button used to bring up the teleport capability will now serve as a laser pointer for UI selection.

- Physics: The embedded physics system can be utilized by the developer for a more immersive experience.

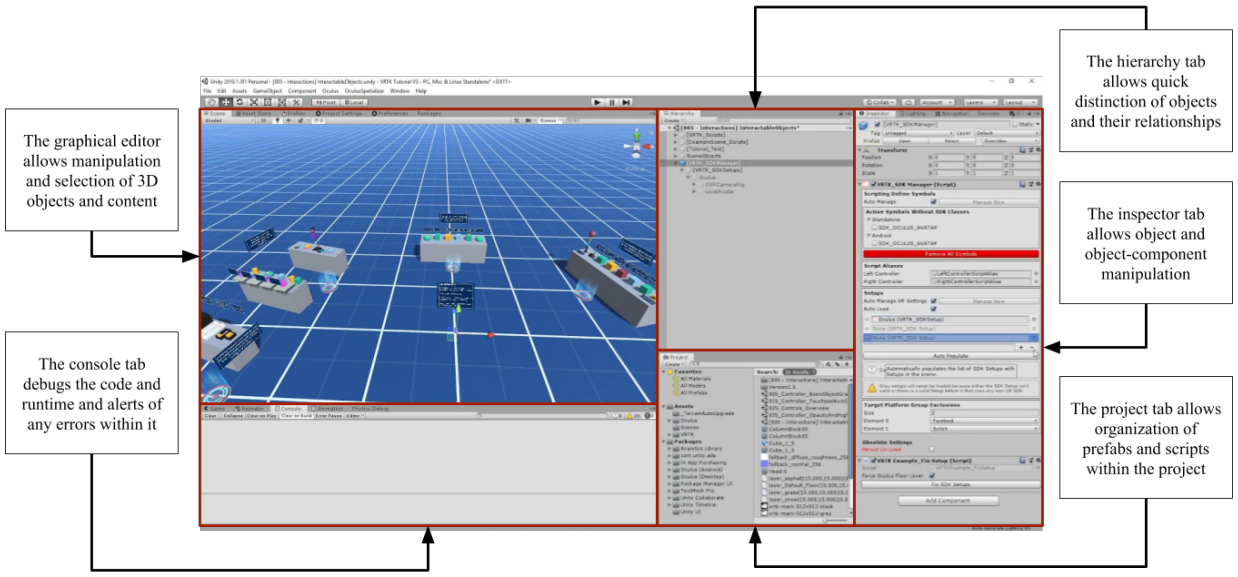


Figure 4.9: The default Unity Editor

The Unity Editor also enables a very visually straightforward development experience, as seen in Figure 4.9 above. Objects can be easily dragged and dropped into the desired location within the Editor and it will show up in the same spot within the visualization. This is much easier to use than the A-Frame method which required defining all objects by $X - Y - Z$ coordinates and troubleshooting if that was not the desired location. Another benefit of Unity is the ability to create pre-fabricated objects (prefabs). Iteratively creating new objects is a demanding process, prefabs

simplifies that process by creating a lightweight copy of the object in memory that users can call and interact with. This simplifies the node creation process as the correct node prefab with all associated properties and components can be called instantly, instead of created and individually compiled at runtime.

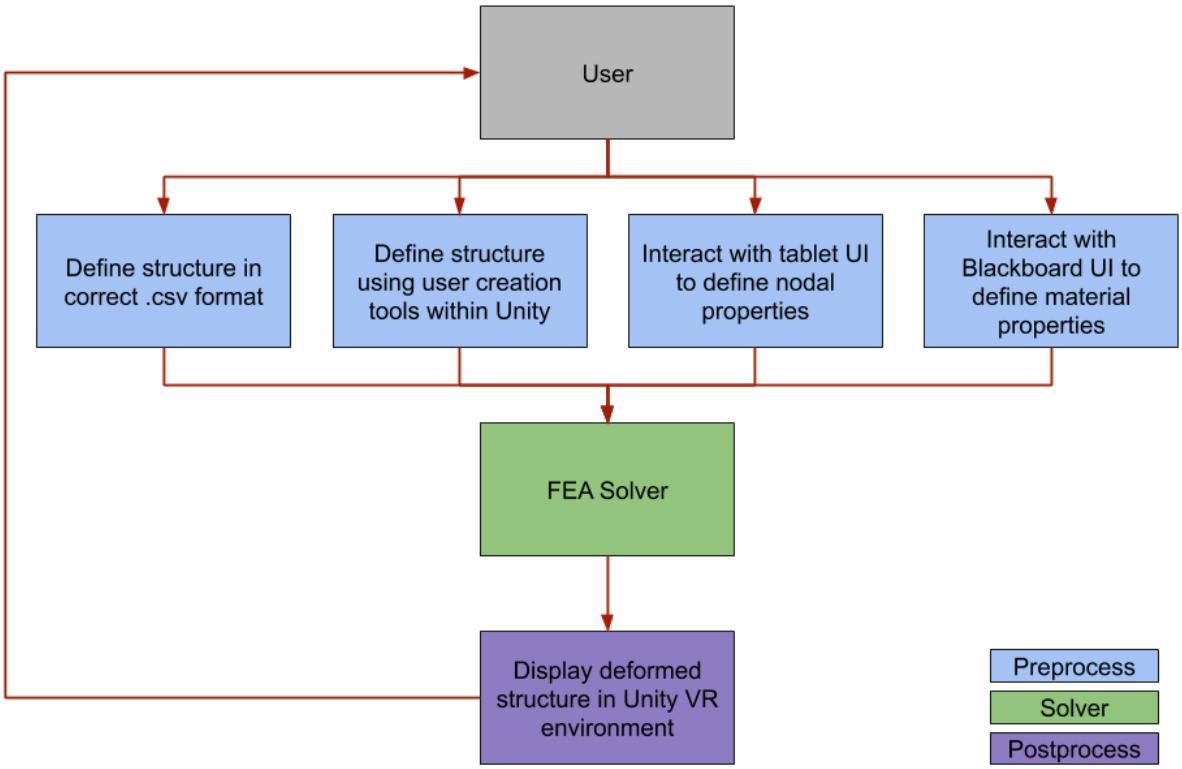


Figure 4.10: Unity user workflow

Because of all the possibilities enabled by the Unity environment, the FEA processing workflow for Unity is a bit different from the A-Frame workflow, as shown above. The user has an extra component in the pre-processing section where they can now define the structure in realtime using the designed creation tools. The traditional .csv structural import is still available as an option for the user. The user can interact with the tablet UI to define properties of each node, such as setting boundary conditions and adding/removing forces. They can also interact with the blackboard UI tool to further manipulate material properties for the structure. The solver then takes in all these user inputs and conducts the structural analysis. The deformed structure is then visualized

in the Unity VR environment so the user can understand what happens to their designs under certain loading conditions. This Unity section will be broken into preprocess, solver and postprocess subsections to individually explain all the steps that go into this process.

4.2.1 Pre-Process

Structure Definition Through File IO

The method for defining a structure is very similar to the on explained above in Section 3.1.1 and shown by Figure 4.2. However, the main difference is that there is no server side to the code in Unity. When the simulation starts, the first script to run is a parser that takes in the data from the .csv file and creates them as node/element objects visually, as well as within code. The C# approach does not use JSON objects like A-Frame did, rather, it uses an object oriented programming (OOP) approach. The nodes and elements are created as class objects from the .csv file. This keeps track of all the components that exist within the structure, and makes future addition and deletion of nodes/elements to the structure a very streamlined process.

Structure Definition Through User Creation

To make new nodes and elements within the simulation, new user tools were designed and implemented. The image below shows the pencil tool which is used to create new nodes and define new elements. The pencil is picked up by the user with the grip button and used with the trigger button, the button mapping is further explained in 4.20. When used, the pencil visually creates a new node at the tip of the pencil object and instantiates a node class object within the solver. The user can then attach this node to the structure by selecting it using the pointer. Selection of a node is visualized by that node turning the color green, setting this as the beginning node for the new element. The user then selects an end node and the element is instantiated between those two nodes visually and an element class object is defined within the solver. The first node now turns back to the initial white color while the end node is now shown green. This enables rapid element definition, as users can quickly make their way across nodes with the old end node serving as the new beginning node for the next element. The eraser tool deletes any nodes/elements that

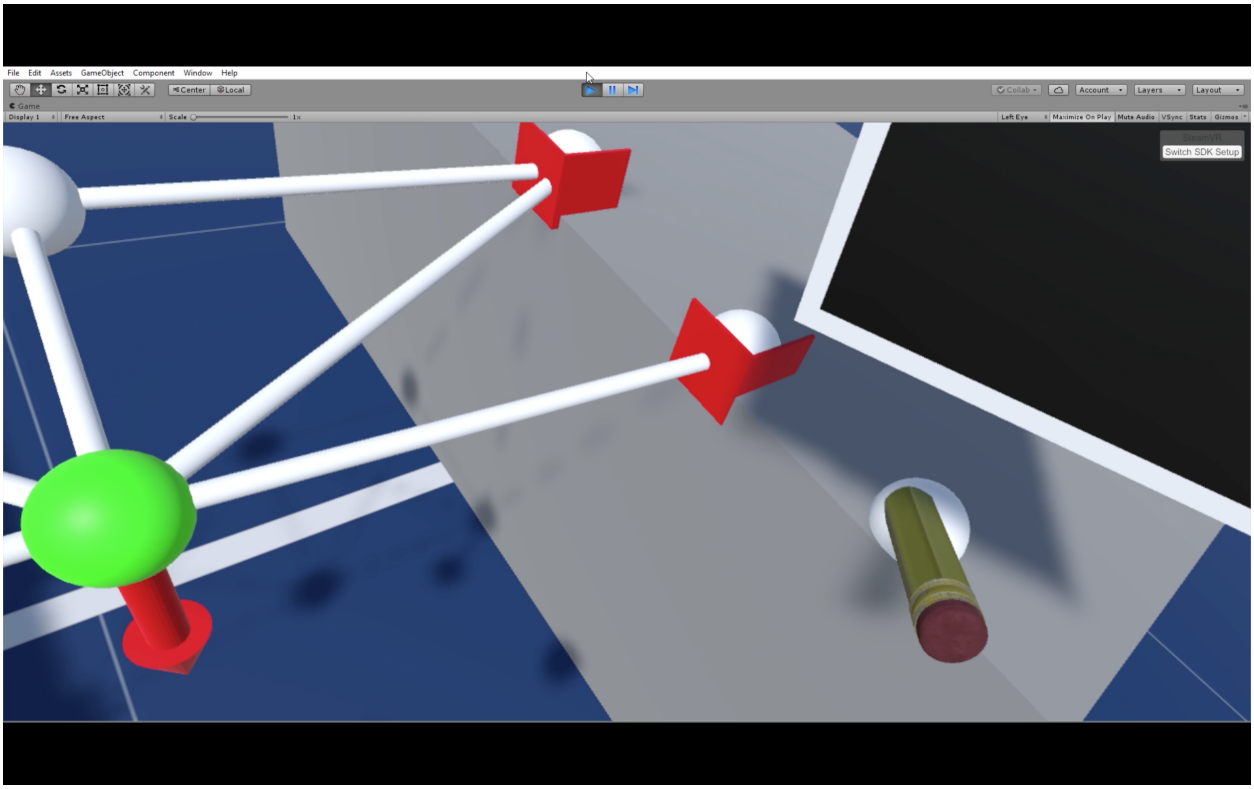


Figure 4.11: Demonstration of the creation pencil

it collides with in the scene. This is enabled by the use of the Unity RigidBody and Physics components. After it visually deletes the object, the eraser tool also erases any references to it within the OOP class system in the Solver.

Virtual Reality User Interfaces

For any existing node, properties for that node can be set through the use of the node tablet. This is a portable tablet that the user can carry around within the scene and interact with using the controller. When a node is selected, it turns green and the node number is populated at the top of the tablet so the user knows which node they are manipulating. The user can then manipulate the boundary conditions of the translation of the node in $X - Y - Z$ directions using the boundary condition toggles. The rotational degrees of freedom can also be manipulated but this functionality is not utilized within the scope of this work and have been left free to rotate. The translational boundary conditions are visualized as red support plates as shown below. When a node is fixed

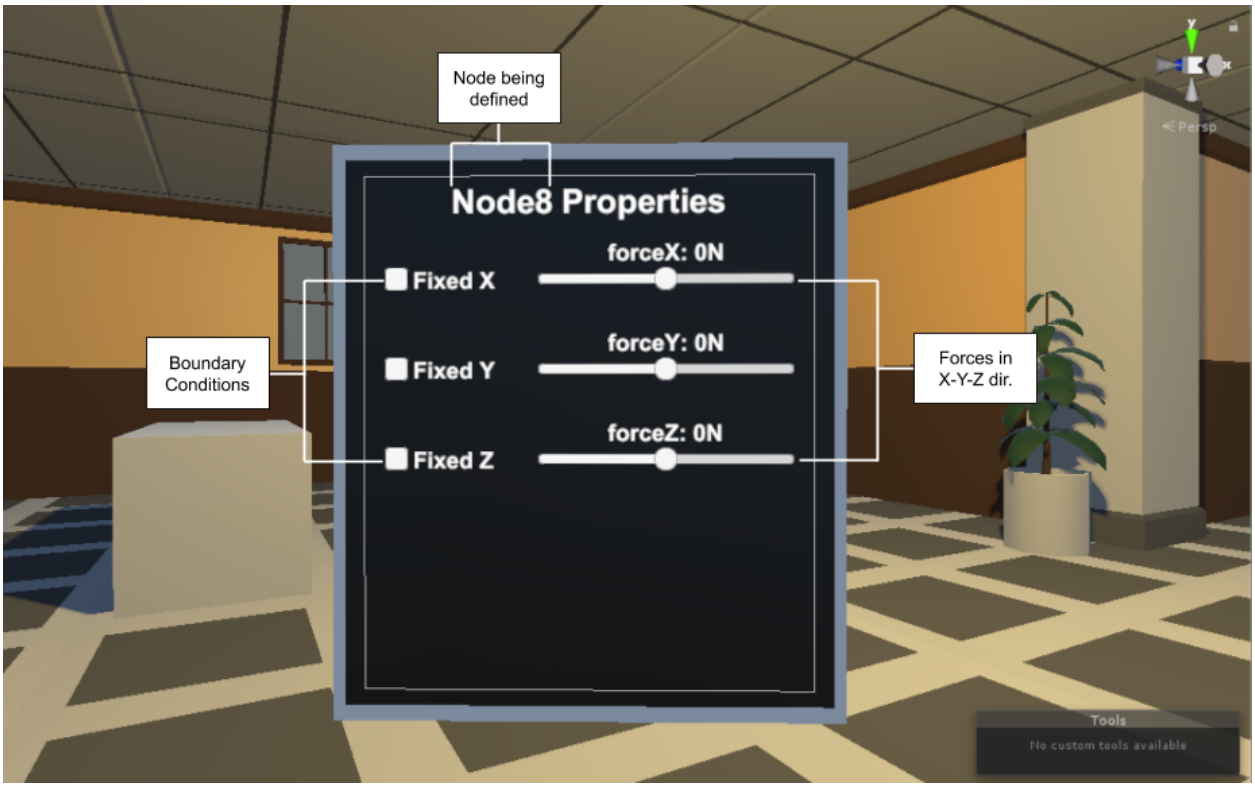


Figure 4.12: Demonstration of the node tablet

in any specific direction, the force slider for that node is then disabled as this would break the simulation.

Boundary condition in **x** direction

Boundary condition in **y** direction

Boundary condition in **z** direction

Boundary condition of all three **xyz** directions

Figure 4.13: Boundary conditions applied to a node in Unity

The forces applied to a node in the $X-Y-Z$ directions are set using a slider on the tablet. This allows the user to experiment with manipulating forces as they deem fit. The forces are visualized using a red force vector with the size of the vector correlating to the magnitude of the force. When forces are applied in multiple directions at a node, a resultant vector is created as shown below.

Force vector in **x** direction | Force vector in **y** direction | Force vector in **z** direction | Resultant force vector of all three **xyz** forces

Figure 4.14: Forces applied to a node in Unity

A Blackboard object was also created within Unity to serve as a focal point for manipulating properties of the structure, as shown in the image below. Some of the functionality enabled by this user interface are:

- Analysis button: Conducts the FEA analysis and visualizes the deformed structure
- Reset button: Resets the structure in the scene to its default configuration
- Young's modulus dropdown: Select between 6 different materials for the structure
- Allowable stress dropdown: Select the stress limit for the structure that the color gradient uses to visualize under-stressed/over-stressed elements.
- Radius slider: Control the thickness of all the elements within the structure
- Scale slider: Control the scale of the structure to fit user needs

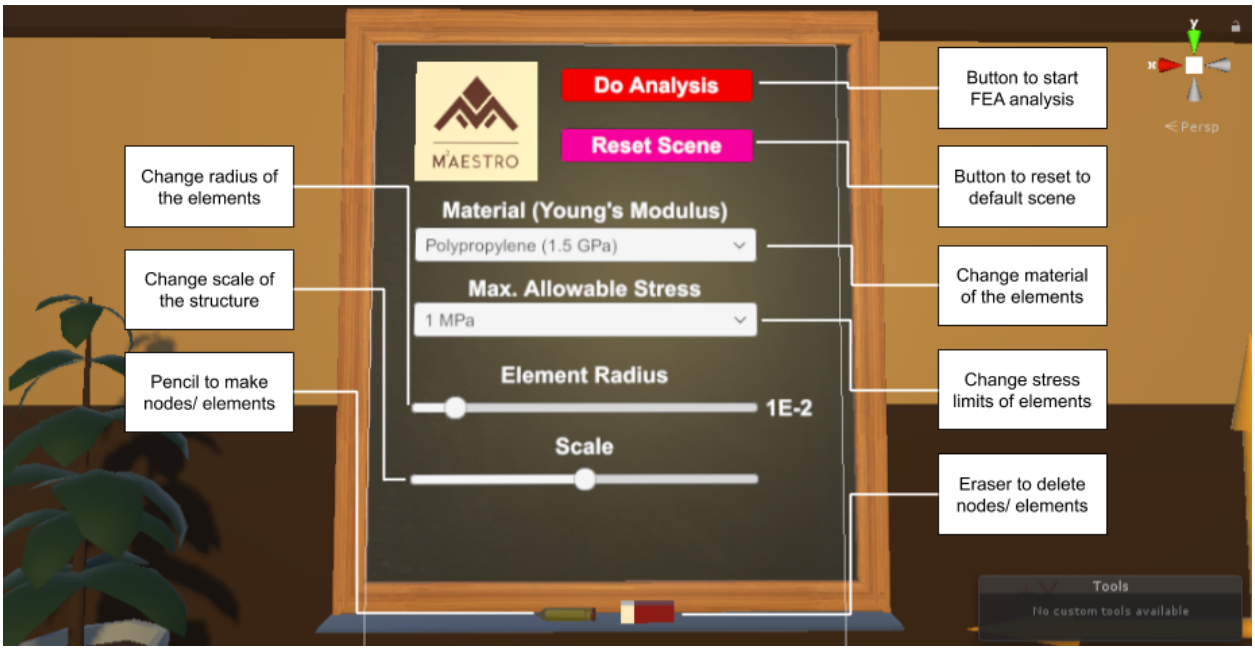


Figure 4.15: Blackboard user interface

4.2.2 Solver

The FEA solver script utilizes the Object Oriented Programming (OOP) approach to node and element classes. The nodes and elements are free to be updated as the user desires, with no immediate impact on the solver, or the deformed structure. Once the user initiates the analysis, the solver finds all the entities that use the node and element class and iteratively process the information using the method from 3.4. The undeformed structure with loading conditions is then hidden, and the deformed structure takes its place. This deformed structure is static and cannot be manipulated, and the user can switch back to the undeformed structure by pressing the analysis button once more to re-enter edit mode. This is shown in the image below.

Figure 4.16: A structure before and after analysis

The $C\#$ implementation of the solver can be viewed in Appendix A. The solver also uses a Cholesky decomposition method for solving the system of equations, rather than the traditional matrix inversion or LU decomposition method. Figure 4.17 below illustrates this further and justifies the use of the Cholesky method within this work. Complexity level 1 has 9 nodes and 14 elements, level 2 has 18 nodes and 28 elements, level 3 has 18 nodes and 44 elements, and level 4 has 27 nodes and 73 elements. The traditional matrix inversion method took 56ms to conduct analysis on the level 4 structure, while the LU method took 20.5ms, and the Cholesky method took 19.5ms at the same complexity level. Methods such as LU and Cholesky Decomposition have been continuously proven to be efficient matrix factorization methods. However, the real key to achieving computational savings comes from knowing beforehand what kind of matrix is being factored and choosing the appropriate algorithm for it. For example, in structural finite element analysis, the matrix being decomposed is always symmetric positive definite [64]. Cholesky decomposition is much more efficient and quicker than LU for those kinds of matrices hence the results in Figure 4.17 below.

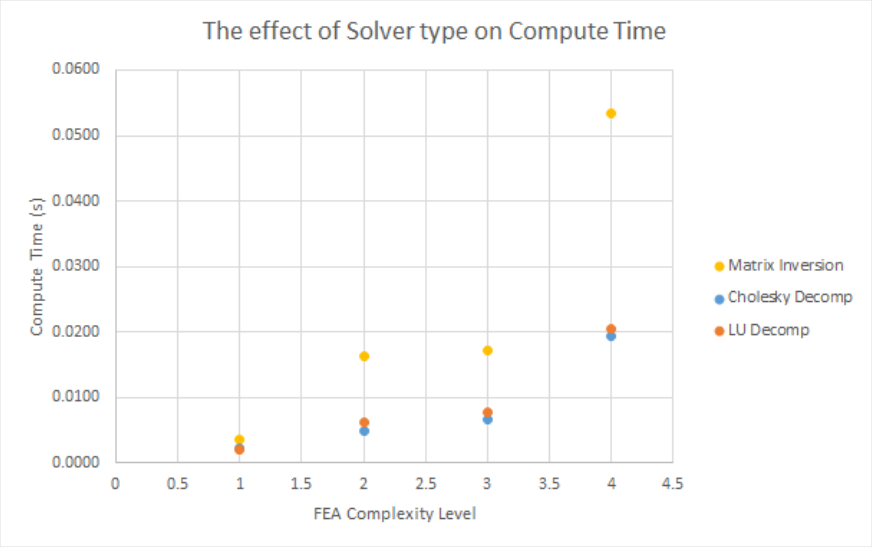


Figure 4.17: Time comparison of FEA solving methods across complexity

To further reduce compute times, other optimization trade studies were researched and conducted. Sparse matrices are a type of matrix traditionally used when the matrix has very few non-zero components. These matrices don't represent the non-zero entries and are therefore faster to analyze. Dense matrices on the other hand have an entry at every possible slot, even if it is mostly zeroes. When both were tested within this script, the dense matrices actually solved faster than the sparse matrix as complexity grew. This is understandable, because as structures start getting more complex and distributing the forces across multiple elements, there will be more non-zero components. As a result, this work uses dense matrices for the FEA solver.

Another trade study was conducted between the usage of floats and doubles in the analysis as these are both data types that exist within the Unity/ $C\#$ environment. Floats are 32-bit floating point data types. Doubles are 64-bit floating point data type with double the precision of a float. The traditional belief is that the more bits used by the data, the longer the computation time will be. Within the scope of the work done in this work, the differences between the two were minimal and Doubles performed slightly faster than Floats. The reason for this is that the $C\#$ linear algebra library is optimized to use Double data types. Doubles are thus the main data type within the solver

as it produces higher precision results at a faster computational speed.

4.2.3 Post-Process

The deformed structure is created by finding the locations of the displaced nodes by adding the displacement $\boldsymbol{D_g}$ to the initial node location and instantiating a prefabricated entity at this new position. The elements are then drawn between the new node locations and given the color corresponding to the stress within. While instantiating straight elements between the deformed nodes of the structure allows its gross motion to be captured, it completely neglects the deformation of the elements between the nodes. This is adequate for truss elements, but the problem arises with frame elements, which are capable of bending. Depicting them with long cylindrical prefabs is an oversimplified visualization. Future work must consider the mathematical form of the element shape functions when determining and rendering their deformed geometric shapes. A prefab is defined as a unique copy of a certain game object, it copies all the properties and components but works and is visualized independently. This allows them to be modular and serve as building blocks for Unity simulations. An example of the element prefab is shown in the image below. In this simulation, many nodes and elements will be created by both the user and through code. It is much less graphically intensive to render a prefab copy for these instances rather than instantiate a new 3D object which would then need to be manipulated to have the correct size, shape and properties.

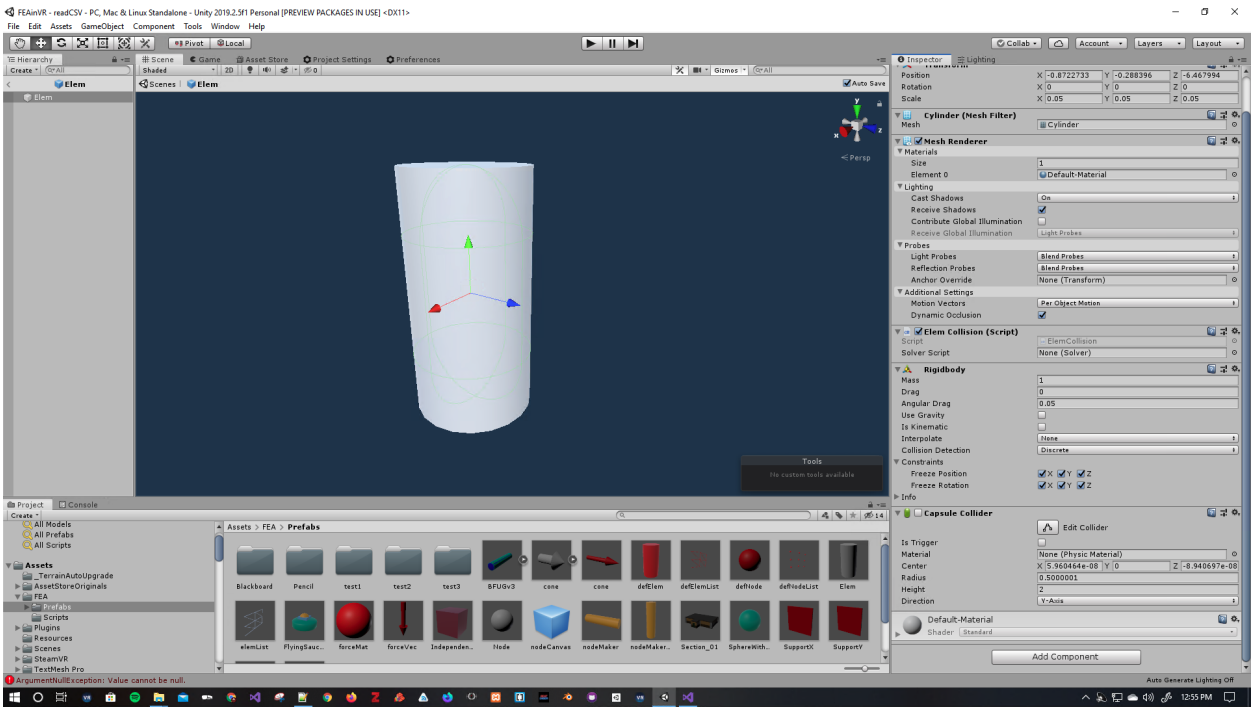


Figure 4.18: The element prefab

These components of these prefabs can also be adjusted by code. These components vary from scale to physics to color. When the deformed structure is created, all the physics interactions are turned off and the nodes get a red color while the elements get a color based on the stress within them. The color gradient for the stress is shown to go from unstressed (blue) to stressed (purple to red) to over-stressed (white), as shown in 4.19 below. This helps the users quickly glance over a structure and see any problem areas for the max allowable stress limit that they set.

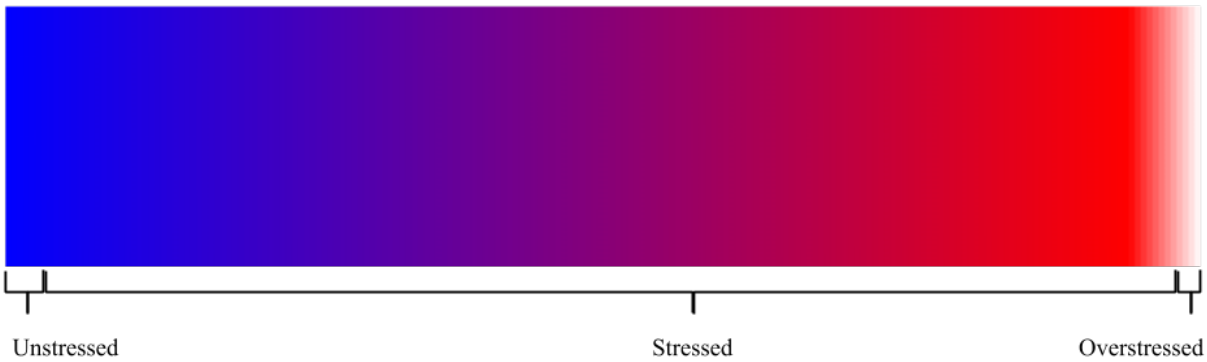


Figure 4.19: Color gradient to visualize stress within elements

VRTK handles the user input and allows for more user interaction capabilities. 4.20 below explains the mapping of actions to the Vive VR controller. The buttons are assigned generically by VRTK, so the trigger on the various other controllers such as Oculus will have the same functionality as the Vive controller. These programmed capabilities enable a higher fidelity VR experience than the A-Frame method. These capabilities will be further utilized and explained in the next chapter.

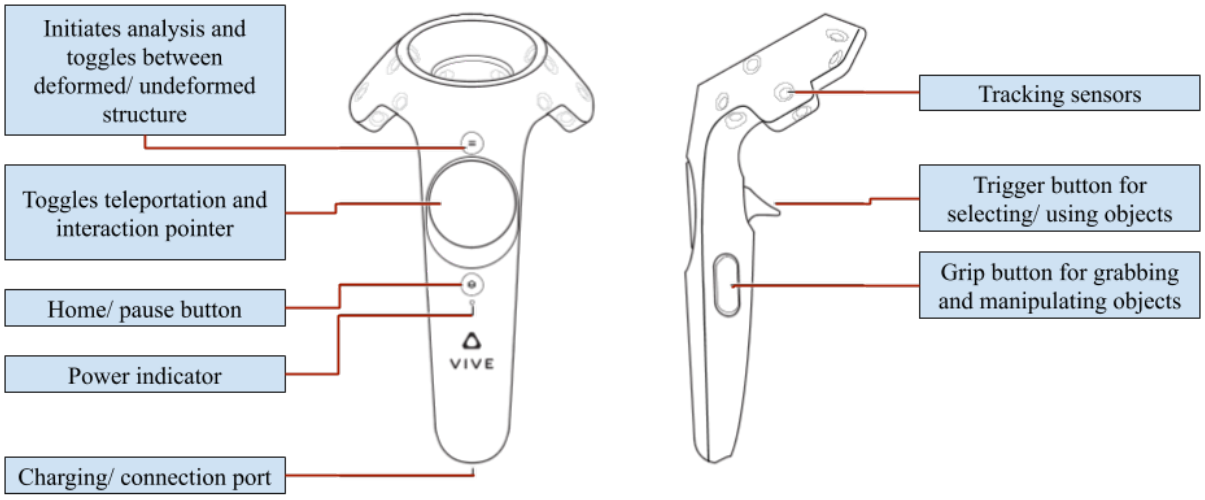


Figure 4.20: Button mapping explained for the HTC Vive controller

4.3 Accuracy of FEA Implementations

Two similar implementations of finite element analysis have been explained in this chapter so far, A-Frame and Unity. However, the accuracy of their analysis has not been investigated. The accuracy of the analysis will be tested against the ABAQUS commercial FEA software. Within ABAQUS, a $Python$ script was written to read through the .csv files defined in the method shown by 4.2. Within ABAQUS, the elements of the structure were defined as 3D deformable wires. the elements were meshed as a B33 element with a seed size of 1 so that no elements are subdivided

into smaller elements in this analysis. B33 represents a 2-node beam element with 6 degrees of freedom in space [65]. This element uses the Euler-Bernoulli beam theory for calculations and is an accurate description of the frame element used in this work. The material of the element is Polypropylene with a Young's modulus of 1.5×10^9 GPa and a Poisson's ratio of 0.43 [66]. The forces are treated as a static concentrated force at a node, and the boundary conditions fix the designated degrees of freedom.

5 structural analysis cases will be compared across A-Frame, Unity and ABAQUS. The accuracy will be determined by the displacement of the structure compared to the ABAQUS analysis. A simple element consisting of 2 nodes will first be analyzed to understand the accuracy across a single element. This element will be first analyzed under translational forces, and then under torsion. The simple element will the be built up into a 2D box structure consisting of 4 nodes and 4 elements for analysis of a simple box structure. This box will then be enhanced further as a simple cube structure consisting of 8 nodes and 14 elements will be analyzed and compared. The comparison will conclude with a more complex tower structure that has 48 nodes and 91 elements.

4.3.1 Single element under translational forces

To validate the accuracy of the implemented solvers, accuracy of a single element was tested first. This element was configured to have a radius of 0.01m with one end encastred (all 6 degrees of freedom fixed) and forces with a magnitude of 10N applied in both X and Y directions of the free end.

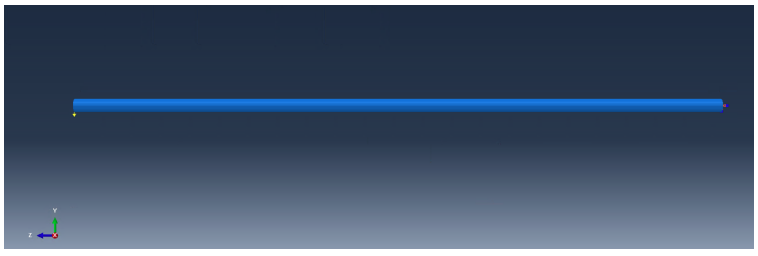


Figure 4.21: Simple element before analysis

The global displacement of the free node was compared across solvers to test for accuracy.

A-Frame and Unity were both able to calculate the translational displacement of the nodes in the structure with a 0.0% error when compared to ABAQUS results. This means that at the single element level, the solvers are completely accurate. The below image shows the three deformed elements across solvers. The structures look near identical with the minute difference of the 3D projection utilized by each system. The A-Frame solver took 8.33 milliseconds to compute this analysis, while the Unity solver took 0.2122 milliseconds.

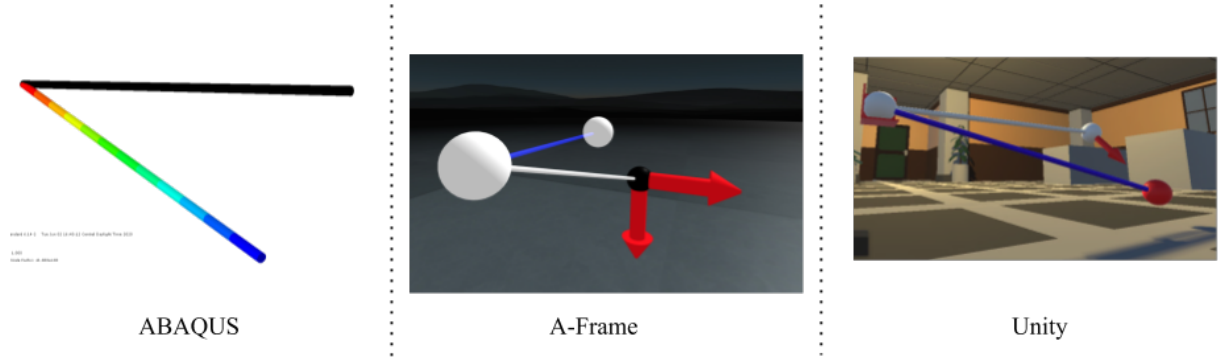


Figure 4.22: Deformed element visualized across solvers

4.3.2 Single element under torsion

The same element as the case above was analyzed, but this time under a moment ($F_{\theta z}$) of 1N. An updated graphic is not shown, as it did not displace the element and would look exactly the same as 4.21. The rotational displacement of the free node was compared across solvers to test for accuracy. A-Frame and Unity were both able to calculate the rotational displacement of the nodes in the structure with a 0.0% error when compared to ABAQUS results. This means that torsion is accurately solved at the element level. The A-Frame solver took 7.65 milliseconds to compute this analysis, while the Unity solver took 0.2251 milliseconds.

4.3.3 2D Box Structure

The analysis was then extended from a single element to a simple 2D box structure. This cube has 2 encastred nodes (restrict all translational and rotational degrees of freedom) with a point

force of -100 Newtons applied in the y-direction of node 3. All elements are Polypropylene and have a radius of 0.01m. An example of this structure modeled within ABAQUS is shown in the image below.

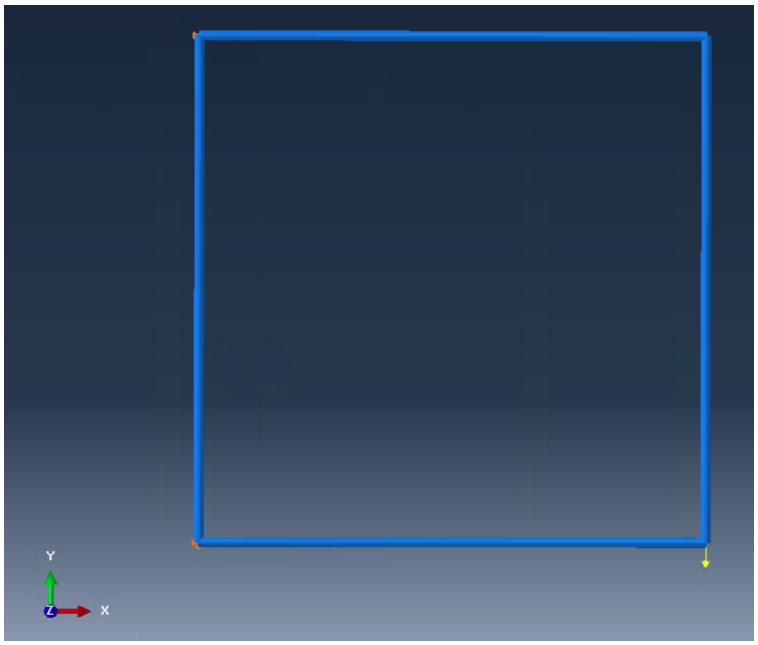


Figure 4.23: Simple element before analysis

The global displacement of the displaced node was compared across solvers to test for accuracy. A-Frame and Unity were both able to calculate the displacement of the nodes in the structure with a 0.0% error when compared to ABAQUS results. This means that at the 2D box structure level, these solvers are completely accurate. The below image shows the three deformed elements across solvers. The structures look near identical with the minute difference of the 3D projection utilized by each system. The A-Frame solver took 8.65 milliseconds to compute this analysis, while the Unity solver took 1.35 milliseconds.

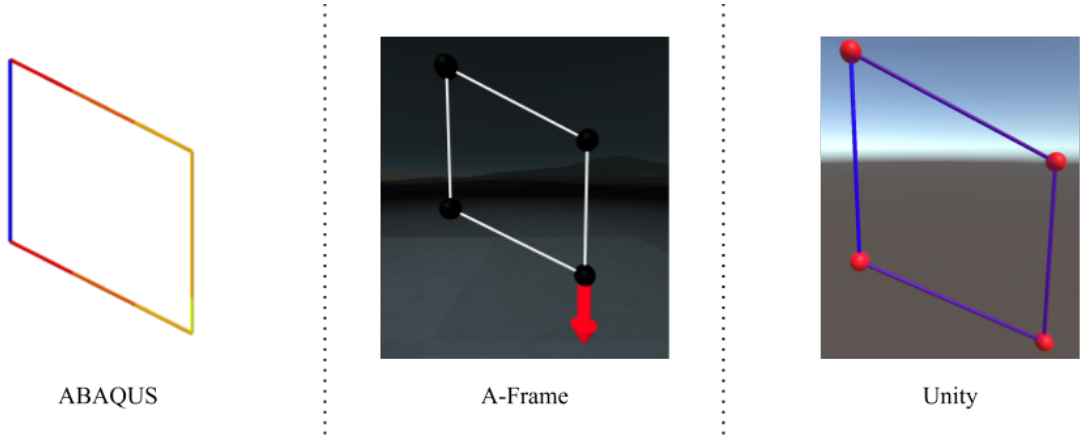


Figure 4.24: Deformed element visualized across solvers

4.3.4 Cube Structure

The analysis was then extended from a 2D box to a 3D cube. This cube has 4 pinned nodes (restrict all translational degrees of freedom) with a point force of -100 Newtons applied in the y-direction of another node. All elements are Polypropylene and have a radius of 0.01m. An example of this structure modeled within ABAQUS is shown in the image below.

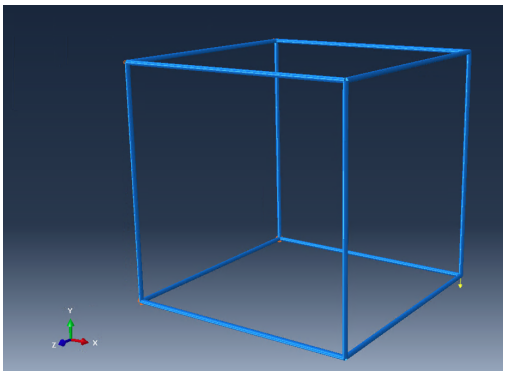


Figure 4.25: Simple structure before analysis

The global displacement of the nodes, $\boldsymbol{D_g}$ was compared across solvers to test for accuracy. A-Frame and Unity were both able to calculate the displacement of the nodes in the structure with a 0.46% error when compared to ABAQUS results. The source of this error is believed to be the

different approaches taken with respect to section analysis by ABAQUS and the virtual reality solvers. The below image shows the three deformed structures across solvers. The structures look near identical with the minute differences of the 3D projections utilized by each system. The A-Frame solver took 14.22 milliseconds to compute this analysis, while the Unity solver took less than 3.15 milliseconds.

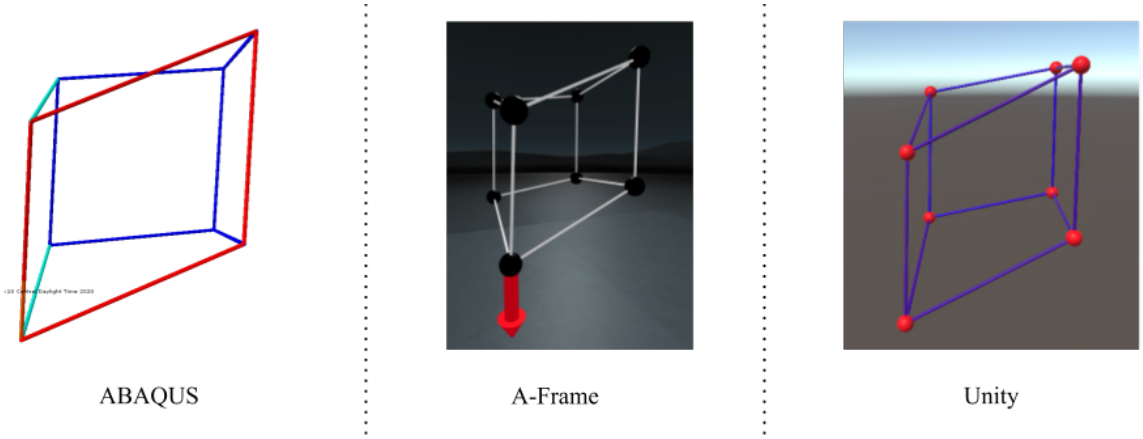


Figure 4.26: Deformed simple structure visualized across solvers

4.3.5 Complex Structure

The complex structure is a miniature tower inspired by an actual physical model. As shown in the image below, this tower was a sufficiently complicated design made by two 4th and 6th grade students using TinkerToy objects. They then wrote the connectivity data of their structure to an Excel document to be read by the solvers. This served as a useful test scenario as the end goal of this work is to be a teaching tool for future structural engineers.

Figure 4.27: Students designing a structure and studying the node positions and connectivity of it

This structure is modeled as having polypropylene elements with radius of 0.01m. The 12 nodes at the bottom have boundary conditions that restrict all translational movement and loading is expressed as a single node having a point force of 100 Newtons applied in the x-direction. An example of this modeled within ABAQUS is shown in the image below.

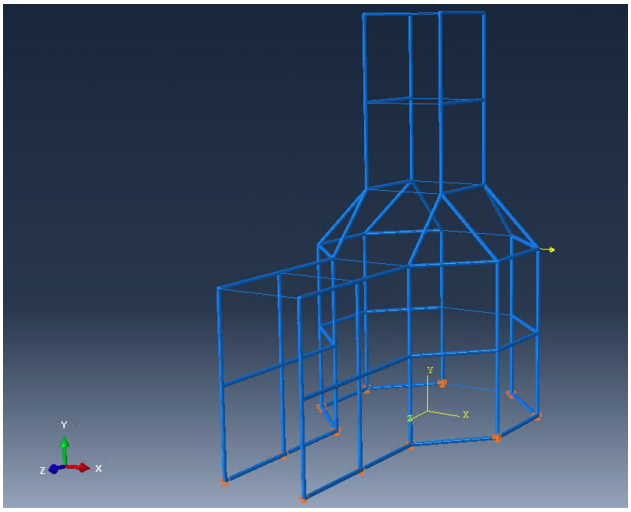


Figure 4.28: Complex structure before analysis

The global displacement of the nodes, $\boldsymbol{D_g}$ was compared across solvers to test for accuracy. Unity was able to calculate the displacement of the nodes in the structure with a percent error of

1.13% when compared to ABAQUS results. A-Frame, however, computed the displacement to a 1.0% error when compared. The difference between these two calculations is believed to be the effect of varying precision. To find area, ABAQUS integrates through the thickness of each element while the implemented solver finds an exact area through a formula. Unity uses Double precision for their calculation, while A-Frame can only use Float (single) precision. This can have a compounding effect as the structure increases in complexity and introduce variations. The below image shows the three deformed structures across solvers. The A-Frame solver took 251 milliseconds to compute this analysis, while the Unity solver took about 12.91 milliseconds.

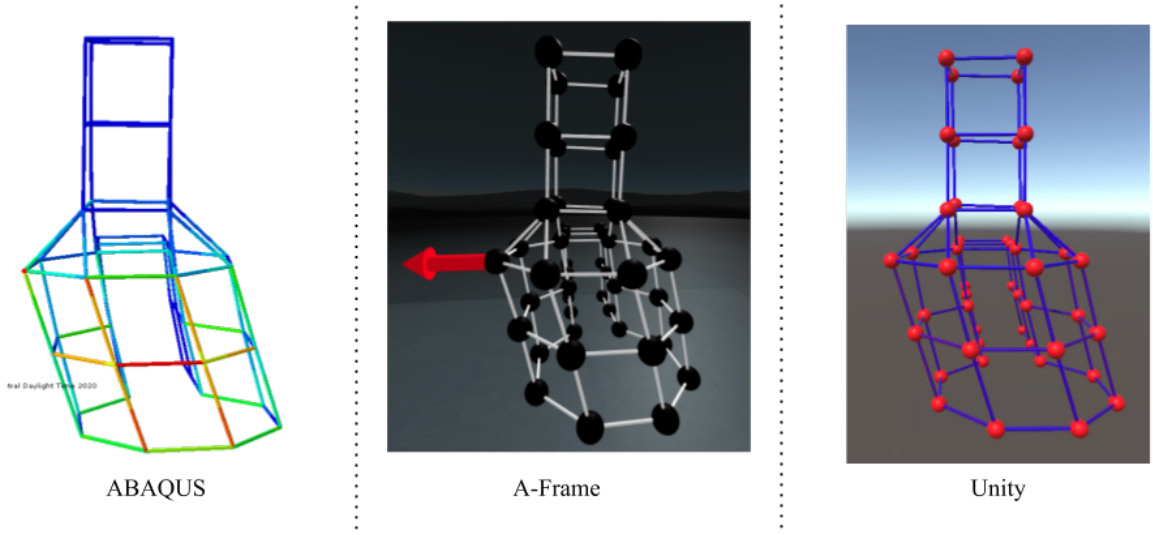


Figure 4.29: Deformed complex structure visualized across solvers

This small inaccuracy between the results of ABAQUS and that of the implemented solver is attributed to the hidden functionality that ABAQUS employs. ABAQUS has a multitude of options that it selects between such as integrating between the thickness of each element to optimize the run time of their solver. The solver used in this work does not have these capabilities and employs a much more direct and simplified approach. There are also a variety of settings and configurations that can be changed to skew the accuracy of these results. This data was compared directly with B33 elements in ABAQUS, however, within that element, there are hybrid formulations and

sectional Poisson's ratio toggles that this solver doesn't use. Manipulating these settings will make the percent error less in some cases but more in others. Considering these options as a source of slight variation in the comparisons, the results from this implemented solver can be believed to be accurate enough. While more rigorous testing can be done to assess accuracy, this data states that the implemented virtual reality solvers can accurately conduct structural analysis with a relatively small percent error.

Shape	A-Frame % Error	A-Frame Runtime	Unity % Error	Unity Runtime
Bending Element	0.0%	8.33 ms	0.0%	0.2122 ms
Torsional Element	0.0%	7.65 ms	0.0%	0.2251 ms
2D Box	0.0%	8.65 ms	0.0%	1.35 ms
3D Cube	0.46%	14.22 ms	0.46%	3.15 ms
3D Tower	1.0%	251 ms	1.13%	12.91 ms

Figure 4.30: Comparison of the accuracy and runtime between the implemented solver and ABAQUS for each case

4.3.6 Visualization of frame elements

The deformed elements shown so far in this work have been rendered as straight cylinders. Straight cylinders are easy to render and accurately represent truss elements, but using them for deformed frame elements can lead to a mischaracterization of the actual deformation of the structure. The displacement of the elements are computed using cubic polynomials as shown by Equation 3.19 for transverse displacements. Because of the higher order polynomials, the elements are usually not a straight line and could be more accurately represented by the use of a spline as shown in Figure 4.31 below. However, creating these splines in Unity using stacked 3D cylinders is not a trivial implementation. The shape of the spline would need to be calculated for each element in the x, y, and z directions using the shape functions, then that spline will need to be rotated from the local coordinate system to fit within the global coordinate system, and then they be rendered within the VR environment. The process for computing and rendering cylinders that are appropriately curved and stretched for each element within the structure would be quite expensive and would ruin the real-time capabilities that this work strives to achieve.

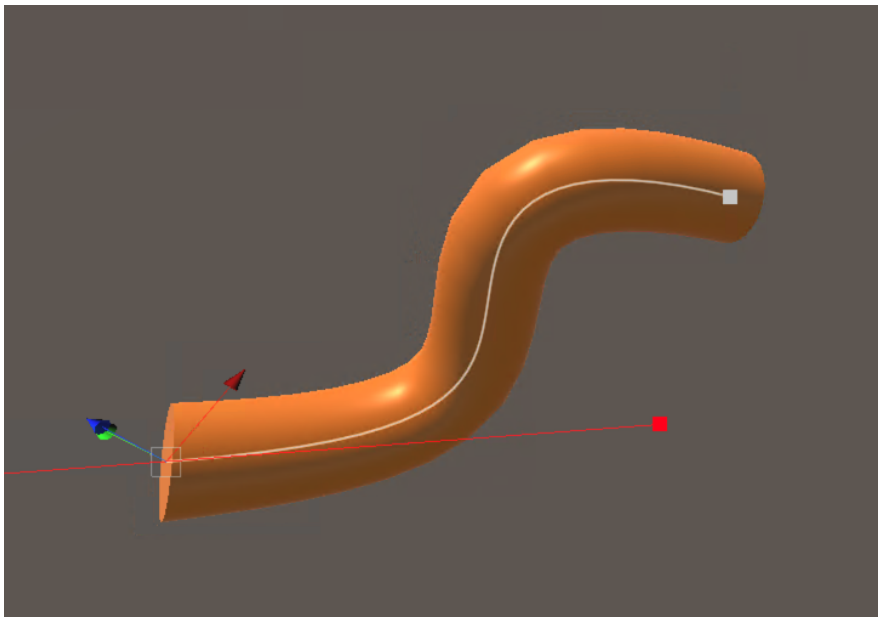


Figure 4.31: Example of a simple spline within the Unity environment

In order to minimize latency and visualization time, this work employs a simple approach to rendering. The correct global displacement of the structural nodes are calculated, and the previously defined element connectivity is used to render a straight cylinder between the nodes of the deformed structure. In order to reduce the complexity of the structure and inherently improve computation speeds, the size of each element is set to be the same size as the structural member being analyzed. While using simple linear elements to represent each member of a frame structure can roughly approximate the structural deformation, it can lead to a deformed visualization which is oversimplified and counterintuitive. The box structure from 4.3.4 above is a good example of this. A possible way to more accurately visualize the deformation of the element without using a polynomial spline is to refine the mesh to use a smaller seed size. With a seed size of 1.0 times the length of the structural member, this coarse box has 4 elements total and its rendered deformations is shown on the left side of Figure 4.32 below. When this box is refined further to a seed size of 0.1 times the length of the structural member, it results in the deformation visualized as the right image in Figure 4.32, which now has 40 elements total.

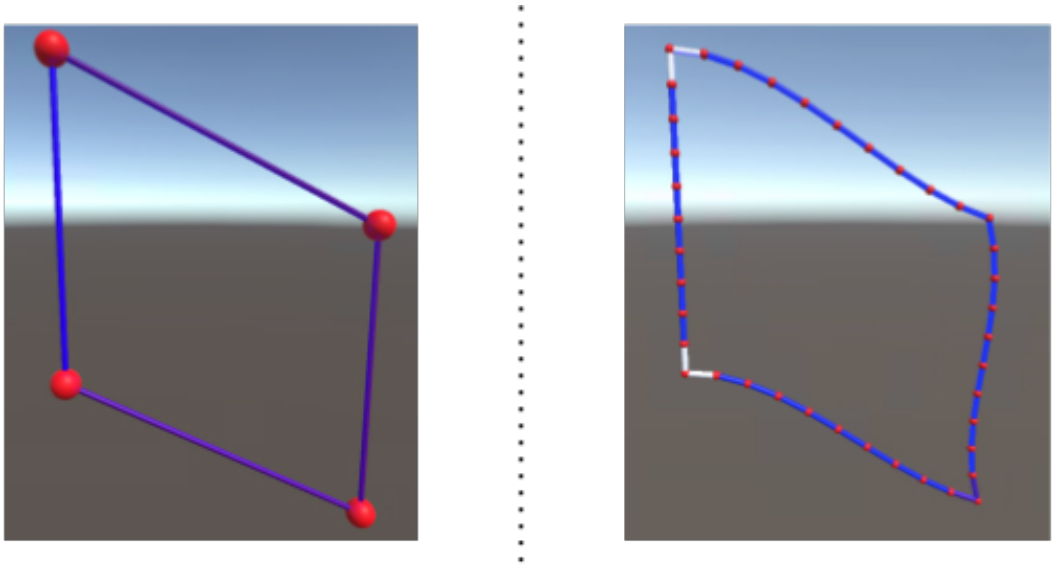


Figure 4.32: Comparison of the same box structure across differing mesh refinement

While these two structures have similar tip displacements that match perfectly with ABAQUS results, their rendered shapes are vastly different. The solution employing the finer mesh exhibits the beam curvature expected of the true solution that are completely ignored by the coarse mesh. The coarse mesh can serve as a rough verification of the analysis but the finer mesh shows a more accurate structural result. Traditionally, when using commercial FEA solvers, the process of mesh refinement is a key step in validating any finite element model and gaining confidence in the results. Meshes are iteratively analyzed with continuously smaller elements until a solution convergence is found where changes between the solutions are minimal and asymptotic behavior of the structure starts to emerge. This is an arduous process which is beyond the scope of this work. In an ideal world, various size meshes could be analyzed in real-time within a virtual reality environment. However, as seen by the visualization speed study in Appendix B, increasing structural complexity causes high latency within the virtual reality simulation. In order to keep simulations smooth, this work will only employ coarse meshes for its analysis and will need to be further refined by future work.

5. IMMERSIVE EDUCATION AND STRUCTURAL DESIGN CASE STUDIES

This chapter will discuss potential educational applications of this work and thus represents the primary demonstration of the contributions of this work. The use cases defined here are intended to be captive to different audiences from young students to professionals but do not encompass all the future avenues for this work. The goal of each case is to provide the intended user with an immersive environment that reinforces and extends their current understanding of structures and structural analysis in some way. Each case will have a virtual environment specifically tailored to their audience which should enhance their sense of presence in virtual reality. Various techniques from literature will also be employed to enhance these experiences. Insight gleaned from the various preliminary VR development efforts of Chapter 2 are itemized below and were considered throughout the case study design process as they provide valuable input in making simulations more immersive and user friendly.

- Scene design is a nuanced task in that it provides the users with a sense of presence and should be tailored to the audience.
- Users benefit from being smaller than the designs so that they could move through them and visualize from both the inside and the outside. This was especially important in complex bodies textured with technical field data.
- Users preferred leaving the model static, observing it and experiencing it in its own context, and would rather move their own bodies instead of moving the model.
- Users were interested in the capability to visualize stress contours; it is important that a color scheme for stress that is easily distinguishable from a glance be implemented in this work.
- Users discussed preferring the virtual navigation technique of teleporting through the VR simulation over "flying" through it. Teleportation provides a more controlled and direct navigational experience and should be utilized in these scenes.

5.1 K-12 Virtual Reality Bridge Design Competition

Designing a bridge has been a common design exercise to enhance the engineering knowledge of K-12 students. These bridges can be designed with toothpicks and marshmallows, building toys like Legos and Lincoln Logs, or a variety of other building items. This task is often performed in the context of a competition with a specific time limit and a small amount of available resources. The structures built by the students are usually tested through the use of various weights to determine which structure from among a set was the strongest. This is a great engineering outreach exercise for K-12 students as they get real-world experience in structural design. However, the competition is greatly limited by the availability of resources and the results can be inaccurate due to a variety of sources, such as brittle toothpicks, stale marshmallows, and the difficulty of assembling any system, structural or not, with precision if one is not experienced. These are some of the aspects addressed by software-based teaching tools such as West Point Bridge Designer.

West Point Bridge Designer is a software that enables students to design and analyze structures using a computer [67], as shown in Figure 5.1 below. This software enables students to rapidly prototype building structures across various constraints such as load and cost without being limited to the construction component options (e.g. length, thickness and materials of members) they have on hand. Students can design bridge structures from their homes and even participate in a national competition. While this software is impressive and engaging, the drawback is that students lose the kinetic intuition that comes from designing the structure by hand. The work in this thesis can extend the capabilities of this software further through a fun and immersive virtual reality experience that enhances the structural engineering knowledge of the students. While these bridges are analyzed using frame elements and the visualizations of their deformed shapes may not be completely accurate, it can roughly approximate the motion of the deformed structure and that should suffice to satisfy the needs of the K-12 audience. Truss analysis would lead to more accurately rendered results and may better match the "toothpick-marshmallow" design scenario, but its instability can lead to student frustration and thus failure of the immersive experience.

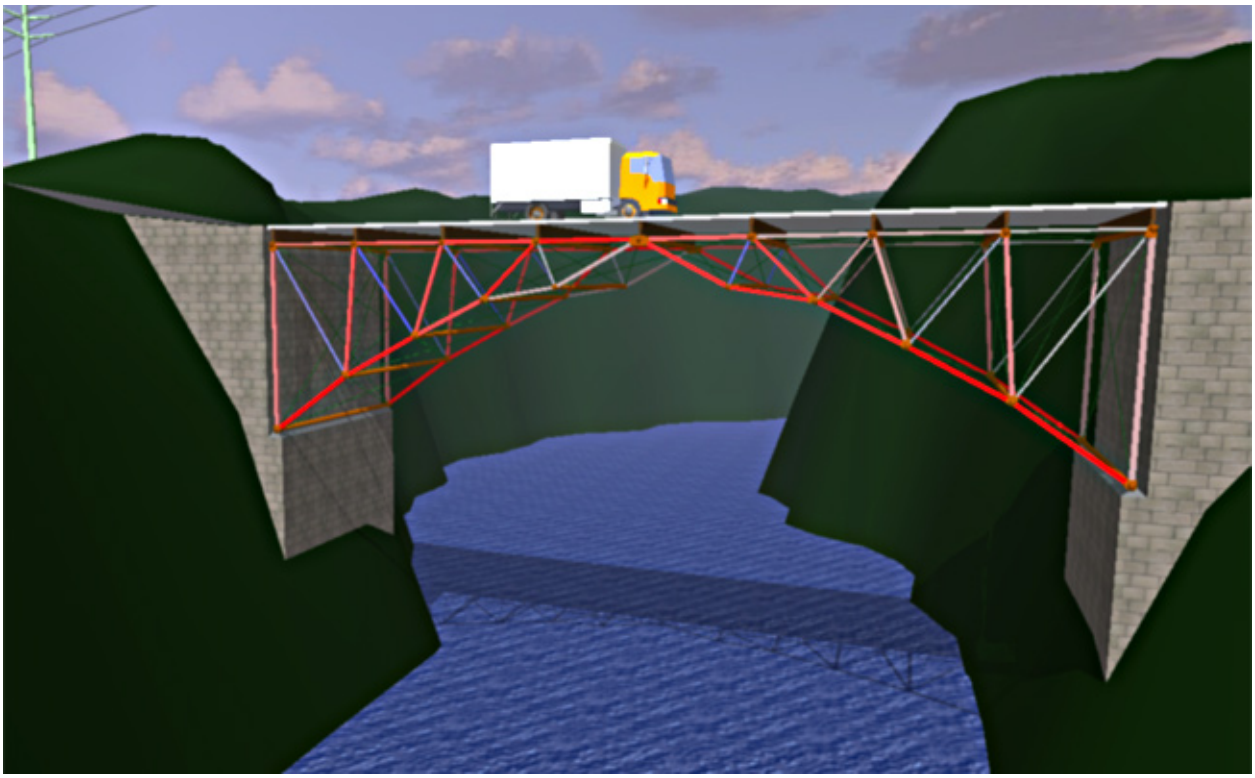


Figure 5.1: West Point Bridge Designer

5.1.1 Educational Objective

The goal of this experience is to immerse students in a virtual reality simulation that challenges their problem solving skills and enhances their structural engineering knowledge. In the scenario used in this case, there are two distant lands separated by a body of water. One land is barren of food and the other land has mushrooms that they could use for sustenance. The design challenge as shown below is to build a bridge that will allow the transportation of mushrooms between the lands.

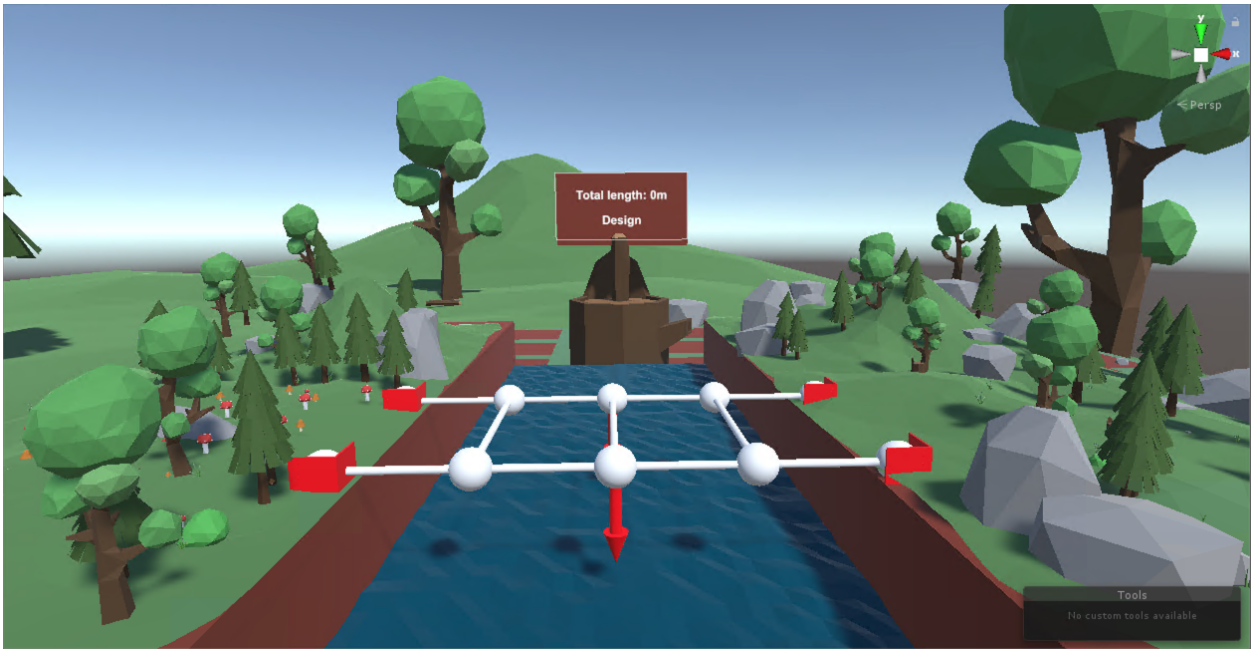


Figure 5.2: Initial bridge design with boundary conditions

This VR experience is designed to be brought into a classroom of students where the initial challenge and how to use the VR system is explained to them. They are then allowed one minute each to put on the headset and experiment with the system in order to understand how they would use it. Each student is then given a couple of minutes to draw by hand the ideal bridge that they believe would be able to hold the maximum amount of weight. They then get one additional minute to put on the VR headset and make that ideal bridge using the VR tools. At the end of their minute, their bridge is tested to see how many kilograms their bridge can hold before it deflects past the allowed limit.

5.1.2 Engineering Design Problem

The engineering challenge is to design a bridge across the water that minimizes the deflection and total linear length of materials used while maximizing the weight that a specific point on the bridge can hold. An example of a basic scenario of this is shown in 5.2 above. This scenario uses kilograms to visualize force as it is an easier to understand concept for students rather than Newtons (1Kg equals 9.8067N of force). Once the timer ticks to zero and the structure is tested,

the length used and maximum weight capability of their structure can be written down as a leader board to select a winner at the end of the competition.

5.1.3 Scene Design

The components in the design of this scene are selected to appease the audience and be simplistic in order to not require heavy computation. The scene objects are royalty free prefab components provided by Unity. These range from trees to mushrooms to water, etc. The water uses an unique tessellation effect to simulate waves within the scenario. A VR approach from literature called World-in-miniature (WIM) is also used here. The WIM world is the small scene in front of the user with scaled down trees and rocks while in the background the full size trees and rocks exist. This is a technique that improves the user's sense of presence and helps them understand the brevity and real world impact of their virtual decisions [32]. Low-resolution poly objects were used to give the scene a cartoon style feel that the intended demographic (K-12) would find appealing. Furthermore, the user tools such as the pencil and tablet are designed to be intuitive to use by reducing the number of clicks necessary to perform an action. This is beneficial as some of the students may have never used a VR headset before and they should be able to easily participate in this experience as well.

5.1.4 Example Results

Once the students put on their headset, the initial design is shown as seen in 5.2 above. Once that structure is analyzed for the first time with a load of 100kg (50kg at each of the two middle nodes), the following result will be shown. The user will know that this is a bad bridge design due to the sign in the background.

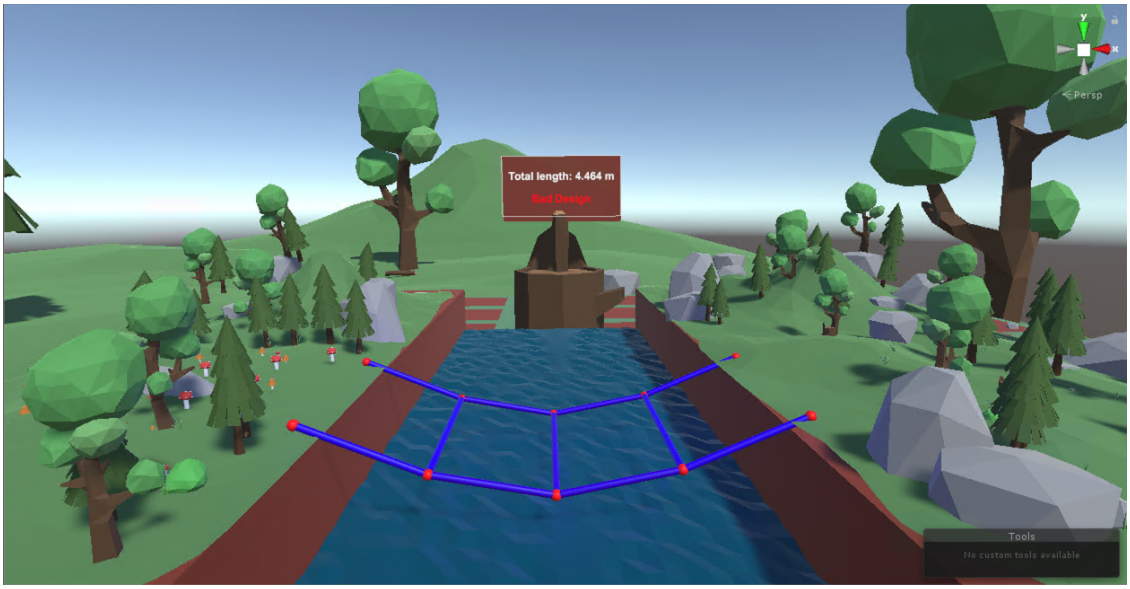


Figure 5.3: Initial bridge design under a 100kg load

The student will then attempt to make this bridge better using the VR experience and an example bridge shown in Figure 5.4 could be created. Analyzing this bridge will update the total length of materials used and whether or not it passes the design criteria to make it a good bridge.

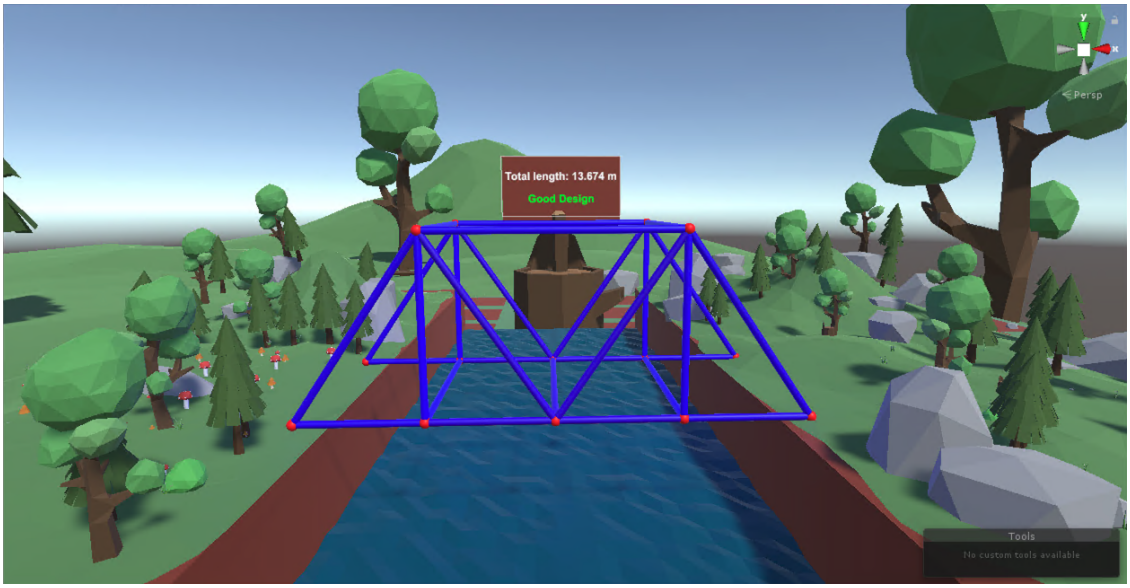


Figure 5.4: Better bridge design under a 100kg load

The amount of weight that this bridge can hold is then continuously tested, until it no longer passes the design criteria as shown by Figure 5.5. The maximum allowed weight is written down along with the total linear length of the bridge's elements as that user's score for the design competition.

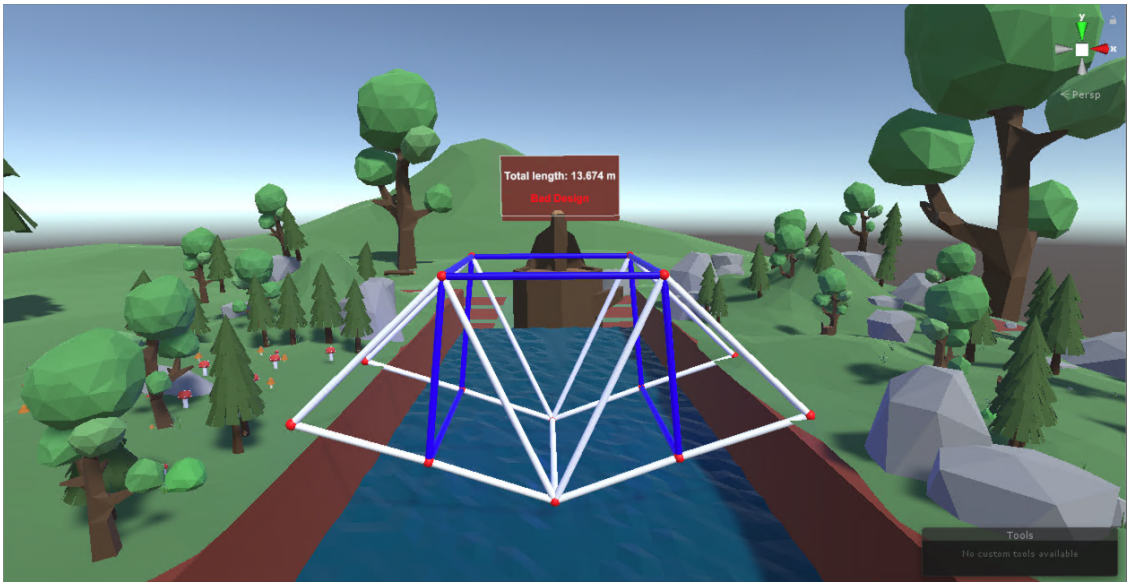


Figure 5.5: Better bridge design under a 10,000kg load

The user can then also go immerse themselves in the bridge that they designed from taking on a miniature perspective as seen in Figure 5.6. This could help develop the student's sense of accomplishment, as well as structural design intuition.

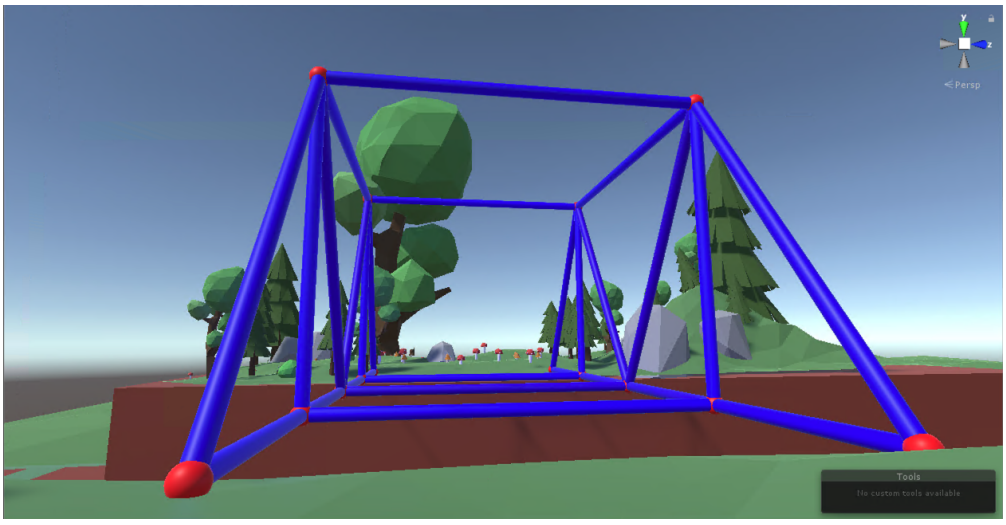
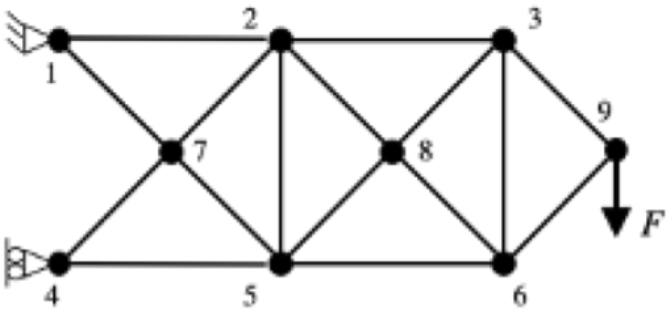


Figure 5.6: User view of bridge from a WIM perspective

5.2 Structural Reconfiguration Challenge

This is a design challenge intended for university engineering students studying structural analysis and the finite element method. This is specifically designed as a supplemental activity for students taking AERO 306 at Texas A&M University. As part of their course, they are required to write a 2D FEA solver to read in input files and conduct analysis of the structure defined there. This challenge takes their assignment one step further by asking them to reconfigure the internal geometry of the structure shown below in order to minimize the maximum stress faced by any one element within the structure.



E = 210 GPa, A = 0.05m², I = 1/12, F = 9.8067N

Node	X-Position	Y-Position
1	1.0	0.0
2	1.0	0.0
3	2.0	0.0
4	1.0	-1.0
5	1.0	-1.0
6	2.0	-1.0
7	0.5	-0.5
8	1.5	-0.5
9	2.5	-0.5

Figure 5.7: Reference structure

5.2.1 Educational Objective

This is a multi-part challenge for AERO 306 students. First they will have to code a FEA solver using the programming language of their choosing to analyze the structure shown in Figure 5.7 above. Their individual solvers will use 2D planar truss elements which is an extension of the bar element explained in section 3.1 to displace in both X and Y directions. They can then use an optimization algorithm to reconfigure the structure to minimize tip deflection, or they can

reconfigure by hand through the trial and error method.

The students will then be given 90 seconds with the VR simulation shown below to manually reconfigure the geometry of the structure to minimize tip deflection while meeting stress and buckling constraint. This will help the students kinetically understand how a structure deforms under a load, and help them get a more intuitive understanding of structural analysis.

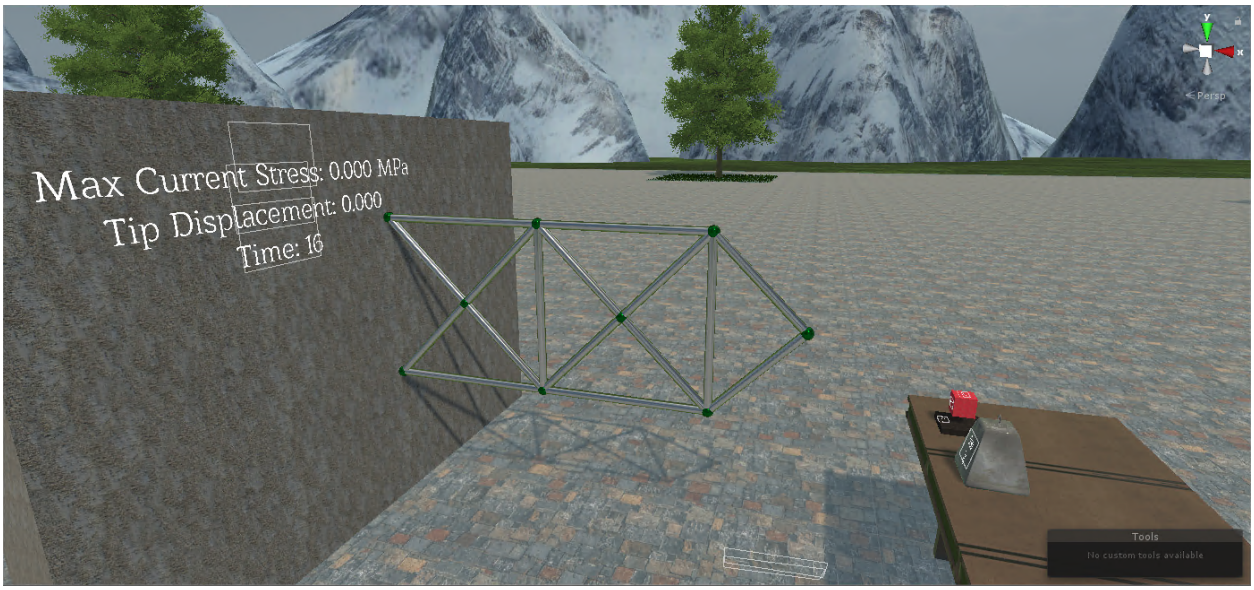


Figure 5.8: Initial structure in VR

5.2.2 Engineering Design Problem

The design problem is formulated in Figure 5.7 above. It can be seen that node 1 is fixed in the X and Y directions, and that node 4 is fixed in the X direction. These are some additional constraints for the VR reconfiguration challenge:

- Node 9 is not fixed, but its location cannot be changed by the user.
- The structure must carry the full load of 9.8067N.
- Additional boundary conditions cannot be defined.

- No element can violate the Euler buckling criteria ($\sigma < \frac{\pi^2 EI}{L^2}$) [68]
- No element can have a stress greater than 22 MPa.

The design challenge is to consider all of the above constraints and reconfigure the initial structure shown below in Figure 5.9 to minimize tip deflection while meeting stress and buckling constraints.

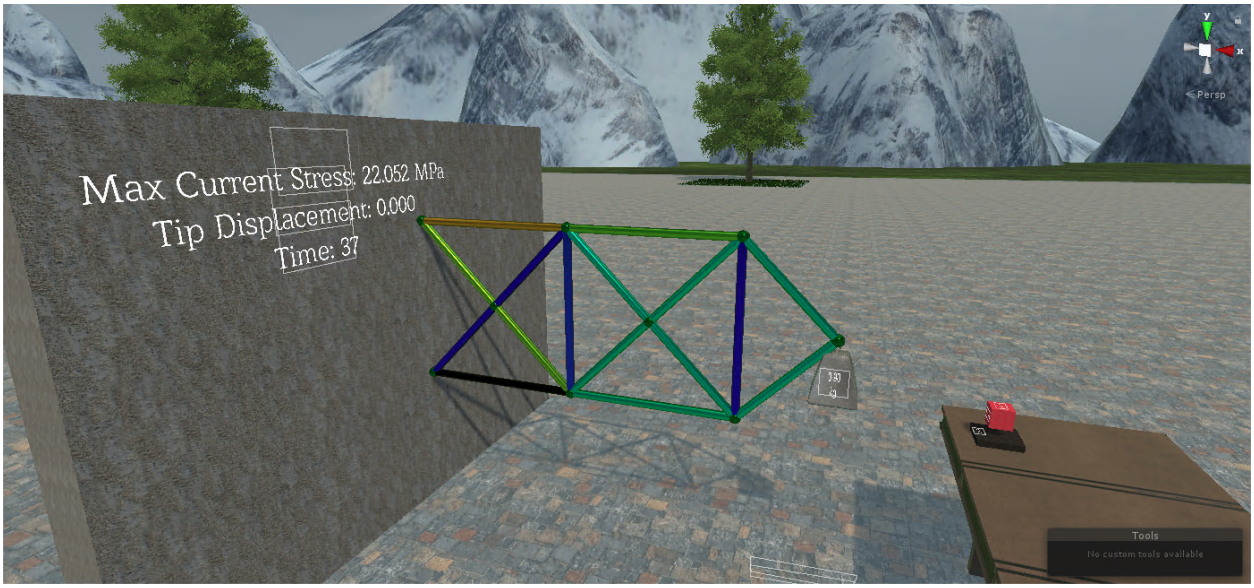


Figure 5.9: Initial structure analyzed

5.2.3 Scene Design

This scene was designed to be very simplistic. The only goal of this simulation is to allow the users to reconfigure the truss structure. The background of the scene is a simple Unity environment with serene mountains and trees scattered throughout. The structure is shown in a real-world configuration attached to a wall and the 9.8067N of force is visualized as a 1kg weight. The nodes are also fixed to not be move able in the Z direction because that would change the fidelity of these results. Once a node is moved, the analysis is conducted immediately and the new deformed configuration is shown. The stress colors within the elements are a unique gradient set by the

magnitude of the axial stress between 0 and the stress limit of 22MPa and are defined as shown in the table below.

Element Color	Magnitude of axial stress
	$\sigma > \pi^2 EI / L^2$
	$\sigma < 4.4$ MPa
	4.4 MPa $< \sigma <$ 8.8 MPa
	8.8 MPa $< \sigma <$ 13.2 MPa
	13.2 MPa $< \sigma <$ 17.6 MPa
	17.6 MPa $< \sigma <$ 22 MPa
	22 MPa $< \sigma$

Figure 5.10: Table associating element stress to color

5.2.4 Example Results

Each student will have an unique solution to this problem as getting the nodes to the exact coordinates they want will be a pretty difficult task. Figure 5.11 below shows an example of an optimized truss configuration in virtual reality. This truss design challenge aims to rebuild the excitement for structures that students had when they were younger playing with lego toys through an academic assignment. It aims to make them think creatively about structural design optimization and increase their intuition for how structures deform.

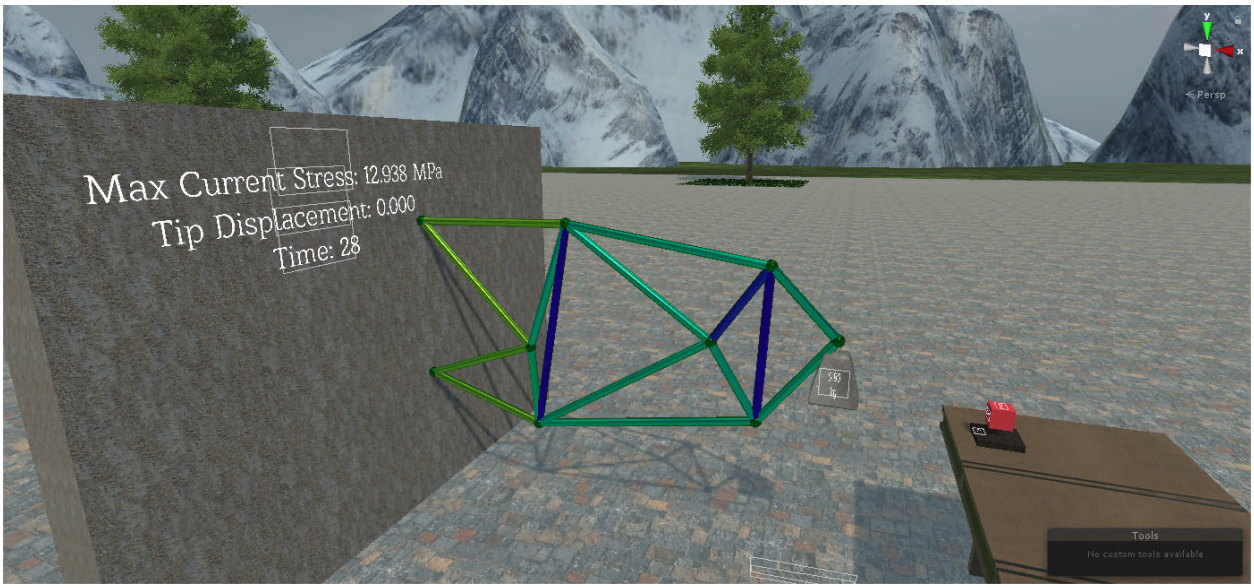


Figure 5.11: Optimized structure analyzed

5.3 Medical Stent Design Collaboration

Coronary stents are small medical devices that are placed in the coronary artery of the heart in order to combat heart disease. As seen by Figure 5.14, the buildup of plaque causes the arteries to narrow, limiting the flow of blood. The coronary stents open these arteries back up when they are deployed. However, these devices are minuscule in scale, with an average diameter of 2.89 millimeters. This makes visualizing these structures a very difficult process. Designing these stents require both medical and structural expertise, however, current design tools do not enable a collaborative design approach [69].

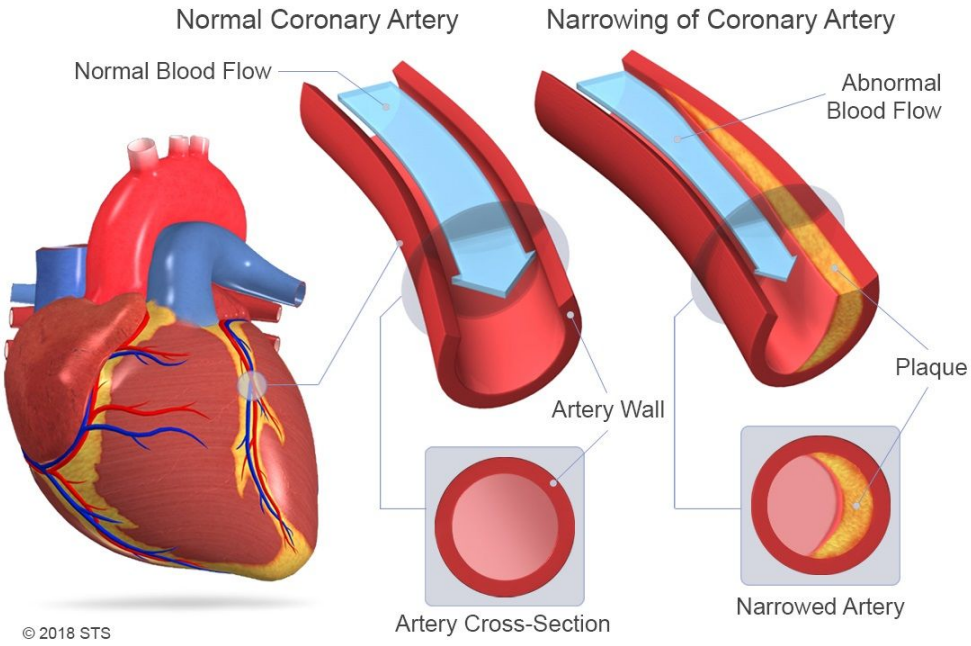


Figure 5.12: The need for coronary stents

5.3.1 Professional Collaborative Objective

The work done in this thesis can be applied to this problem by creating a collaborative tool for highly educated non-structural medical experts to visualize and manipulate the stents that they are helping design. While the visualizations of the deformed elements within the stent may not be completely accurate, this tool would allow users to observe meaningful quantitative results of

the analysis. Metrics such as maximum stress within the stent members and reactive force applied to the artery by the stent are more important in driving their design exploration than are the local deformed shapes of individual elements. They can then rapidly prototype possible stent designs alongside the structural engineer to create a device that meets their objectives for size, safety and cost.

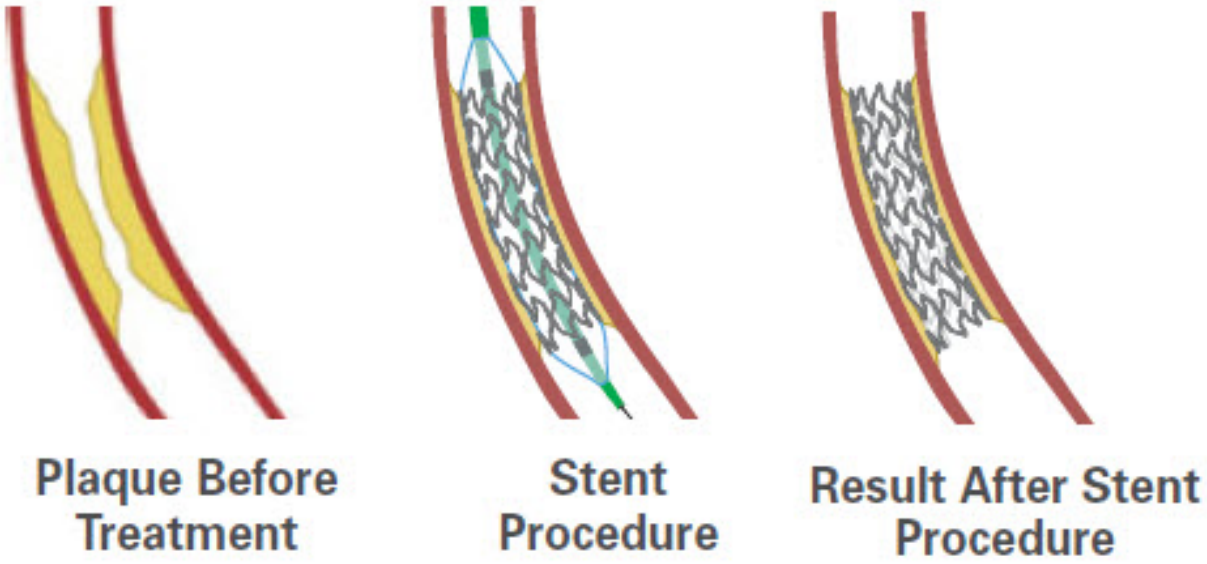


Figure 5.13: Stent in action

5.3.2 Engineering Design Problem

Due to the geometrical imprecision that exists with user interaction in virtual reality, it is not recommended that the stents be designed in the VR environment. Rather, the VR simulation exists as a supplemental tool that will allow the professionals to immersively experience their stent designs and collaborate on design changes and topology. The problem defined here is an expanded stent under the pulsile kinematic boundary conditions of a tapered coronary artery as it contracts and expands. To implement this scenario within the solver explained previously in this work, some changes to the solver capabilities had to be made first. When the solver starts applying boundary

conditions, there is no longer just the conditions of the node being fixed or free, now a force is able to be applied at that node location. This is done through the use of a technique known as the penalty method [60]. The simple method that was used previously only accounted for fixed nodes by zeroing out the related nodes and columns of the $\boldsymbol{K_g}$ matrix, and by putting a 1 at the diagonal to keep the matrix non-singular. However, with the penalty method, the kinematic boundary condition is multiplied by a large number (10^{15}) and set as the $\boldsymbol{F_g}$ force vector, and the $\boldsymbol{K_g}$ value is replaced by 10^{15}. A pseudo-code is shown below that should replace Algorithm 4 in the generic FEA solver implementation.

Algorithm 7 Applying kinematic boundary conditions

$\hat{\boldsymbol{K_g}} = \boldsymbol{K_g}$

$\hat{\boldsymbol{F_g}} = \boldsymbol{F_g}$

for `i = 0; i < n; i++` **do**

  for `b = 0; b < 6; b++` **do**

    if $\boldsymbol{BC_i}[b] ==$ `NOT FREE` **then**

      $\hat{\boldsymbol{K_g}}[6i + b,\ 6i + b] = 10^{15}$

      $\hat{\boldsymbol{F_g}}[6i + b] = BC_i * 10^{15}$

    end if

  end for

end for

`Solve for displacement(`$\boldsymbol{D_g}$`) via:` $\hat{\boldsymbol{F_g}} = \hat{\boldsymbol{K_g}}\boldsymbol{D_g}$

`Solve for reactionary forces (`$\boldsymbol{F_g}$`) via:` $\boldsymbol{F_g} = \boldsymbol{K_g}\boldsymbol{D_g}$

In this specific scenario, the stent is being designed for a tapered artery. As a result, the magnitude of the kinematic boundary conditions on the nodes vary based on its z location within the tapered artery. A simple diagram explaining this is shown below. The goal of this simulation is to have the engineer and medical professional work together to change the internal configuration of the stent and reduce the stress within all the elements of the stent without inflicting a painfully

high reactionary force on the artery internals.

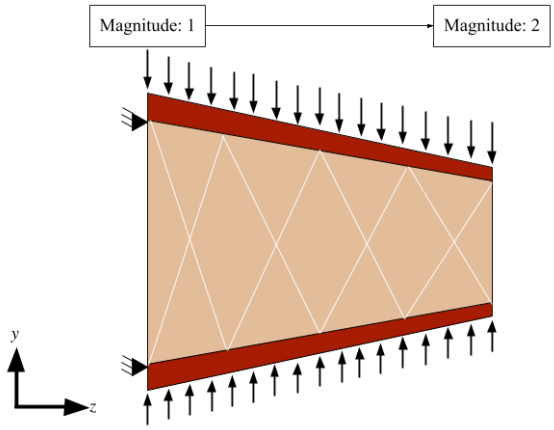


Figure 5.14: Tapered artery design problem

5.3.3 Scene Design

The scenario for this scene is set inside a doctor's office to bring about a sense of familiarity for the intended audience. This was accomplished through the use of royalty free Unity prefabs from the asset store and is visualized below. The hollow tapered cylinder in the scene is intended to approximate the shape and size of a blood artery scaled up a 100 times for visibility.

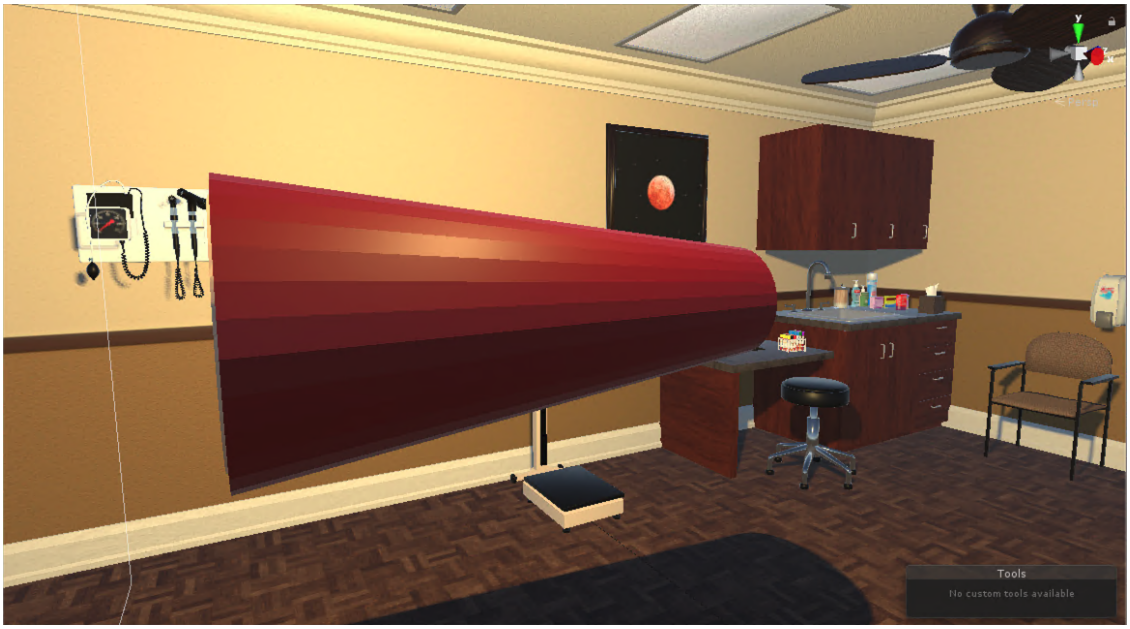


Figure 5.15: Medical scene design

The taper of the artery turns on and off with the structural analysis, this is intended to simulate the pulsile boundary conditions. Figure 5.16 visualizes the initial stent before the analysis is done. The black arrows at each node are indicative of the kinematic boundary conditions there and visualizes which direction and with what magnitude the force is applied.

Figure 5.16: Initial undeformed stent inside untapered artery

Figure 5.17 shows what the stent structure will look like after it is deformed. The stress within the elements is visualized as a blue to red gradient as explained in Figure 4.19 above. There is also an associated point light at each node which changes color to visualize the $F_{reaction}$ at that node and allows the medical professional to quickly denote points of interests at a glance.

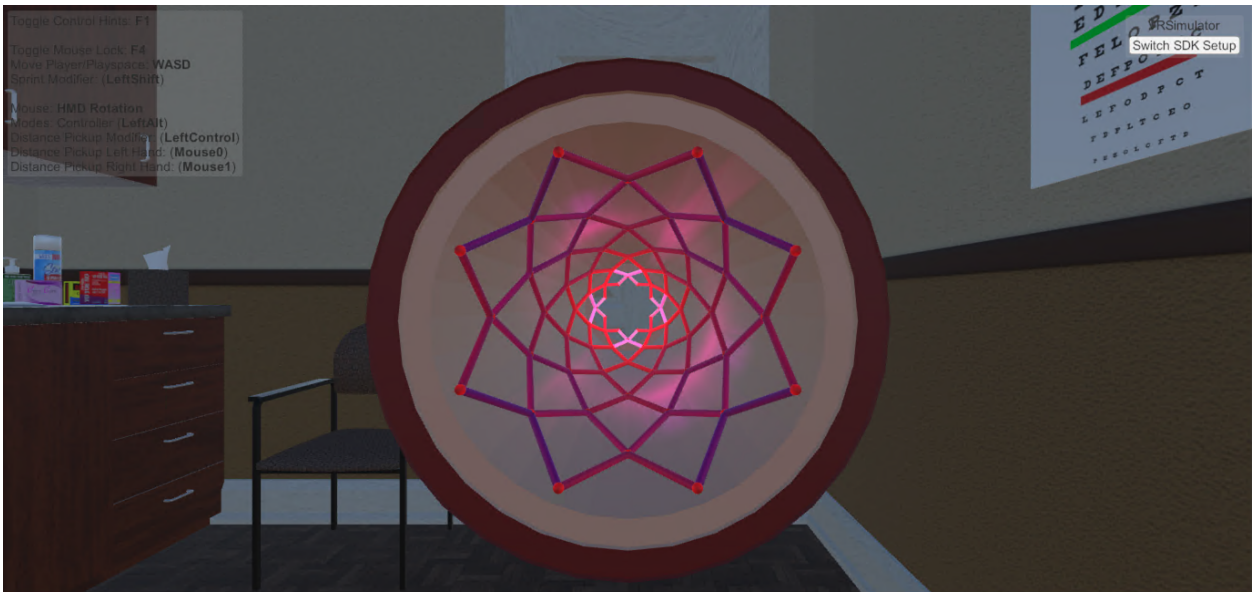


Figure 5.17: Initial stent under kinematic boundary conditions

5.3.4 Example Results

The goal of this scenario is to reconfigure the internal nodes and elements of the structure and completely get rid of any white over-stressed members while still meeting the defined structural criteria. As the medical professional reconfigures the structure, they will get a wide range of results from their modifications, as shown in Figure 5.18.

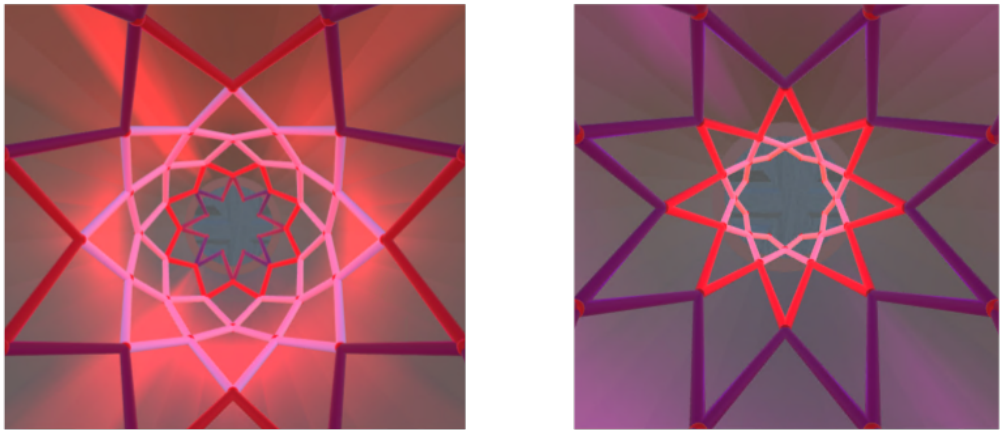


Figure 5.18: Comparison of two stent designs

These changes will help drive their intuition of an ideal stent structure for the scenario, eventually leading them to design one that meets all their criteria and constraints as seen in Figures 5.19 and 5.20.

This stent design activity starts the discussion on what other medical devices and device design processes could benefit from the use of virtual reality applications. Various other medical devices such as casts, 3D printed implants, and bionic limbs could all benefit from a collaborative design process done in virtual reality.

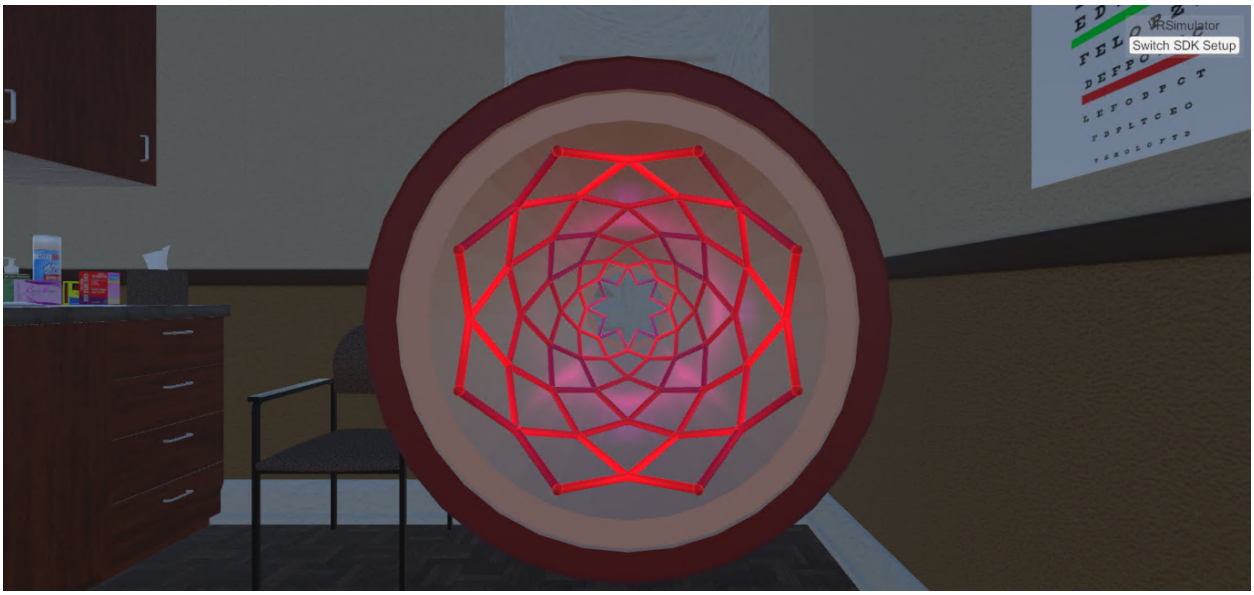


Figure 5.19: Final stent design that meets all criteria

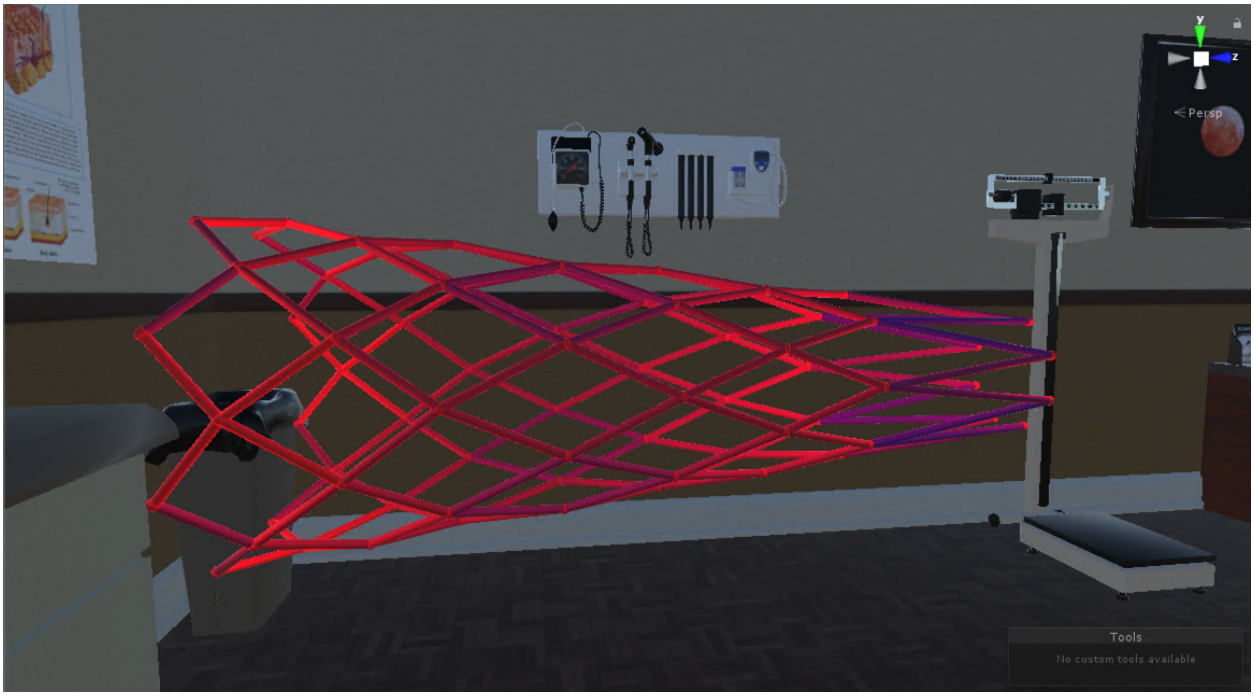


Figure 5.20: Side profile of final stent design

6. CONCLUSIONS AND FUTURE WORK

The goal of this thesis was the development of a software tool to conduct real-time design and analysis of structures within Virtual Reality environments. This goal was accomplished through the implementation of finite element analysis solvers within both an A-Frame and Unity environment. While the visualization implemented in this work is not robust enough to capture the full deformation of the frame elements, the groundwork has been laid for this to be quickly developed in the future. This tool can still be a valuable asset for students and professionals in the structural engineering realm, as it can immerse them within the structures they are analyzing and provide a cost effective alternate to 3D prototyping. The following sections summarize the findings of this work and offer some new ways in which this work can be researched and utilized in the future.

6.1 Conclusions

Chapter 2 overviews the preliminary work done on this project leading up to the work of this thesis. The motivation and need for a realtime FEA tool in virtual reality is discussed and the literature approach of post-processing results from a commercial solver is implemented. Observations from previous projects are discussed leading up to the definition of a lessons learned database that was crucial to the work done in Chapter 5.

Chapter 3 introduces the finite element analysis equations and variables used in this work. The formulation of FEA with 3D frame elements is clearly explained. A generic solver that can solve structures of n nodes and m elements is then introduced and implemented. This chapter builds up the core of the solving capabilities in this work and is expanded and utilized by the remaining chapters.

Chapter 4 discusses how the finite element analysis is implemented within virtual reality environments. Both A-Frame and Unity environments are researched as potential avenues for this work. A-Frame has many positive qualities such as a device agnostic framework and the direct embedding into a website for easy access that Unity does not possess. However, Unity has the

capabilities to solve complex FEA structures in real-time with minimal latency which is the main goal of this work. Unity also has other great qualities such as the ability to utilize developer SDKs to program device specific interactions, the visual GUI which makes scene design very simple, and the ability to make and utilize prefabricated objects that minimize latency and computational overhead. Chapter 3 also tested the accuracy of the FEA implementation of this work with the ABAQUS commercial FEA solver. The findings are that the solver can match ABAQUS results from within a 0.0% to less than 2.0% error. This small inaccuracy is associated with the multitude of options that ABAQUS has and selects between such as integrating between the thickness of each element to optimize the run time of their solver which the solver in this work does not have. Considering that as a source of error in the comparisons, the results from this implemented solver can be believed to be accurate.

Chapter 5 then shifts the focus to explore future use cases for this work. Section 5.1 discusses an implementation for K-12 students that will allow them to explore structural design under limitations. The bridge design challenge closely emulates that made by West Point but immerses the students in their designs in the hopes of providing a more valuable engineering experience. Section 5.2 is intended to aid university students taking a course in structural analysis. The goal is to aid their knowledge development and add to their intuition about how structures deform under loading conditions. Section 5.3 explores a use case for medical design collaboration within virtual reality. A scenario was formed where the user needs to help reconfigure the design of a stent that is supposed to go inside a tapered artery. This section discusses how kinematic boundary conditions can be added to the FEA solver and how the VR scenario was designed. The goal was to immerse the medical professional within the artery that they are designing in order to increase their understanding of the structural integrity of the stent.

This work has accomplished many milestones and has critically developed the concept of a realtime structural analysis tool in virtual reality. Some of the accomplishments achieved through the development of this thesis are:

- A workflow for visualizing SolidWorks models in a virtual reality simulation

- A workflow for visualizing ABAQUS results in a virtual reality simulation
- A collaborative approach to the traditional STEM classroom through the A-Frame environment.
- Various simulation design lessons for creating immersive virtual reality experiences for the classroom.
- A realtime FEA tool that allows users to design and analyze a structure wholly within a virtual reality simulation.
- 3 use cases that can be utilized and extended by students and professionals alike for virtual reality structural analysis purposes.

As a result of the developments within this thesis, the following insights were gained:

- Immersive experience of the kind created lead to positive student feedback (written and verbal), indicating that the learning experience was both valuable and enjoyable.
- Inhabiting the same space as one's design, even virtually, allows accelerated understanding of design features and even critical faults.
- Sufficient computational resources and well functioning off-the-shelf hardware and software options exist that allow structural deformations computed via linear FEA to be rendered without adverse latency in virtual reality environments.

6.2 Future Work

Further continuation of this work has many avenues for growth. Immediate next steps are:

Develop a more robust spline prefab for visualization to replace straight cylinders and more accurately capture the deformations of frame elements. Such a prefab would consider the mathematical form of the element shape functions when determining and rendering their deformed geometric shapes. This omission is one of the key weaknesses of this work and

will need to be enhanced with future implementations.Implement a multi-threaded approach to this work that offloads the computational process and visualizes the deformed structure once the analysis is complete. Network the A-Frame simulation to utilize a server for multi-user FEA simulation. Test Unity simulation across multiple HMD devices (HTC VIVE, Oculus Rift, Valve Index) to ensure compatibility. Utilize a green screen to enhance this experience for students by adding in a mixed reality component. Get IRB approval to test effectiveness of this work on learning comprehension of K-12 students.

A very interesting research study that has yet to be conducted is one on human subjects with IRB approval. A VR simulation such as the one proposed in this work could be tested on students across age groups and interests to gauge how it affects learning comprehension and retention. The results from that study would then drive iterations that would enhance this work as well as the domain of education using virtual reality. The medical stent use case proves the extensibility of this work to fit the design needs of any defined scenario. Future work can be done on different design scenes and challenges that incorporate this work for educational outreach. In a world that is quickly embracing online and virtual education techniques in the aftermath of the coronavirus pandemic, a system that builds on this work could greatly enhance a student's learning experience.

Other avenues for this work are in researching the utilization of real-time force feedback within Unity. Preliminary work has already been done with the MAESTRO lab at Texas A&M in applying real world forces to the nodes within the structure. A user using the designed system or a Novint Falcon system could feel the stiffness of an element and apply a virtual force by displacing a rigged node. This should be developed further into a simulation that lets students and professionals embrace aspects of the physical world within a virtual reality simulation.

REFERENCES

[1] A. Petrovits and A. Canossa, "From M.C. Escher to Mass Effect: impossible spaces and hyper-real worlds in video games. How can hyper-real worlds be designed and interpreted in a 2D, 2.5D and 3D virtual environment and how will this implementation affect the stereoscopic 3D video games of the future?," *G|A|M|E Games as Art, Media, Entertainment*, vol. 1, no. 2, 2013. Publisher: Ludica Section: Journal.

[2] W. K. Li, A. Y. C. Nee, and S. K. Ong, "Mobile augmented reality visualization and collaboration techniques for on-site finite element structural analysis," *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 09, p. 1840001, June 2018.

[3] A. van Dam, D. H. Laidlaw, and R. M. Simpson, "Experiments in Immersive Virtual Reality for Scientific Visualization," *Computers & Graphics*, vol. 26, pp. 535–555, Aug. 2002.

[4] C. Chou, H.-L. Hsu, and Y.-S. Yao, "Construction of a virtual reality learning environment for teaching structural analysis," *Computer Applications in Engineering Education*, vol. 5, no. 4, pp. 223–230, 1997.

[5] J. Huang, S. Ong, and A. Nee, "Real-time finite element structural analysis in augmented reality," *Advances in Engineering Software*, vol. 87, pp. 43–56, Sept. 2015.

[6] T. P. Yeh and J. M. Vance, "Applying Virtual Reality Techniques to Sensitivity-Based Structural Shape Design," *Journal of Mechanical Design*, vol. 120, pp. 612–619, Dec. 1998.

[7] B. Perles, "Interactive Virtual Tools for Manipulating NURBS Surfaces in a Virtual Environment," *Journal of Mechanical Design*, vol. 124, June 2002.

[8] C.-C. P. Chu, T. H. Dani, and R. Gadh, "Evaluation of virtual reality interface for product shape designs," *IIE Transactions*, vol. 30, pp. 629–643, July 1998.

[9] I. Sutherland, "The ultimate display," *Proceedings of the IFIP Congress*, pp. 506–508, 1965.

[10] G. C. Burdea, "Virtual reality technology," 2003.

[11] S. Bryson, "Virtual reality in scientific visualization," *Communications of the ACM*, May 1996.

[12] F. P. Brooks, "What's Real About Virtual Reality?," *IEEE Computer Graphics and Applications*, p. 12, 1999.

[13] H. Regenbrecht, G. Baratoff, and W. Wilke, "Augmented reality projects in the automotive and aerospace industries," *IEEE Computer Graphics and Applications*, vol. 25, no. 6, pp. 48–56, 2005.

[14] A. Liverani, F. Kuester, and B. Hamann, "Towards interactive finite element analysis of shell structures in virtual reality," in *1999 IEEE International Conference on Information Visualization (Cat. No. PR00210)*, (London, UK), pp. 340–346, IEEE Comput. Soc, 1999.

[15] K.-S. Choi, H. Sun, and P.-A. Heng, "An efficient and scalable deformable model for virtual reality-based medical applications," *Artificial Intelligence in Medicine*, vol. 32, pp. 51–69, Sept. 2004.

[16] M. Bro-Nielsen, "Finite element modeling in surgery simulation," *Proceedings of the IEEE*, vol. 86, no. 3, pp. 490–503, 1998.

[17] R. C. Alexander and D. K. Smith, "Fumbling the future : how xerox invented, then ignored, the first personal computer," 1999.

[18] HTC, *The VIVE headset*, 2016.

[19] S. Jayaram, J. Vance, R. Gadh, U. Jayaram, and H. Srinivasan, "Assessment of VR Technology and its Applications to Engineering Problems," *Journal of Computing and Information Science in Engineering*, vol. 1, no. 1, p. 72, 2001.

[20] U. Technologies, *Unity Real-Time Development Engine*, 2014.

[21] P. Fillatreau, J.-Y. Fourquet, R. Le Bolloc'h, S. Cailhol, A. Datas, and B. Puel, "Using virtual reality and 3D industrial numerical models for immersive interactive checklists," *Computers in Industry*, vol. 64, pp. 1253–1262, Dec. 2013.

[22] Mozilla, *Introduction to A-Frame*, 2017.

[23] H. Hedelin, "Design and evaluation of a user interface for a WebVR TV platform developed with A-Frame," *Master's thesis, Linkoping University, Linkoping, Sweden*, 2017.

[24] M. Usoh, K. Arthur, M. C. Whitton, R. Bastos, A. Steed, M. Slater, and F. P. Brooks, "Walking > walking-in-place > flying, in virtual environments," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, (Not Known), pp. 359–364, ACM Press, 1999.

[25] D. A. Bowman and L. F. Hodges, "An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments," in *Proceedings of the 1997 symposium on Interactive 3D graphics - SI3D '97*, (Providence, Rhode Island, United States), pp. 35–ff., ACM Press, 1997.

[26] L. Kohli, M. C. Whitton, and F. P. Brooks, "Redirected Touching: Training and adaptation in warped virtual spaces," in *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, (Orlando, FL), pp. 79–86, IEEE, Mar. 2013.

[27] J. S. Pierce and R. Pausch, "Comparing Voodoo Dolls and HOMER: Exploring the Importance of Feedback in Virtual Environments," p. 8.

[28] J. Goble, K. Hinckley, R. Pausch, J. Snell, and N. Kassell, "Two-handed spatial interface tools for neurosurgical planning," *Computer*, vol. 28, pp. 20–26, July 1995.

[29] A. Steed, S. Frlston, M. M. Lopez, J. Drummond, Y. Pan, and D. Swapp, "An 'In the Wild' Experiment on Presence and Embodiment using Consumer Virtual Reality Equipment," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, pp. 1406–1414, Apr. 2016.

[30] E. Klein, J. Swan, G. Schmidt, M. Livingston, and O. Staadt, "Measurement Protocols for Medium-Field Distance Perception in Large-Screen Immersive Displays," in *2009 IEEE Virtual Reality Conference*, (Lafayette, LA), pp. 107–113, IEEE, Mar. 2009. ISSN: 1087-8270.

[31] M. Billinghurst and H. Kato, "Collaborative augmented reality," *Communications of the ACM*, vol. 45, July 2002.

[32] S. Richir, P. Fuchs, D. Lourdeaux, D. Millet, C. Buche, and R. Querrec, "How to design compelling Virtual Reality or Augmented Reality experience," 2015.

[33] E. M. Kolasinski, "Simulator sickness in virtual environments," *Technical Report 1027, United States Army Research Institute for the Behavioral and Social Sciences*, May 1995.

[34] S. Scherer and M. Wabner, "Advanced visualization for finite elements analysis in virtual reality environments," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 2, pp. 169–173, Aug. 2008.

[35] E.-J. Lee and S. El-Tawil, "Femvrml: An interactive virtual environment for visualization of finite element simulation results," *Advances in Engineering Software*, vol. 39, no. 9, pp. 737 – 742, 2008.

[36] M. Connell and O. Tullberg, "A framework for the interactive investigation of finite element simulations in virtual environments," in *Developments in engineering computational technology*, pp. 23–28, GBR: Civil-Comp press, Dec. 2000.

[37] M. J. Ryken and J. M. Vance, "Applying virtual reality techniques to the interactive stress analysis of a tractor lift arm," *Finite Elements in Analysis and Design*, vol. 35, pp. 141–155, May 2000.

[38] E. G. Parker and J. F. O'Brien, "Real-time deformation and fracture in a game environment," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 156–166, Aug. 2009.

[39] T. W. Sederberg and S. R. Parry, "Free-form deformation of solid geometric models," in *Proceedings of ACM SIGGRAPH 1986*, pp. 151–160, Aug. 1986.

[40] W. Li, A. Nee, and S. Ong, "A State-of-the-Art Review of Augmented Reality in Engineering Analysis and Simulation," *Multimodal Technologies and Interaction*, vol. 1, p. 17, Sept. 2017.

[41] D. Marinkovic and M. Zehn, "Survey of Finite Element Method-Based Real-Time Simulations," *Applied Sciences*, vol. 9, p. 2775, Jan. 2019. Number: 14 Publisher: Multidisciplinary Digital Publishing Institute.

[42] R. Hambli, A. Chamekh, and H. Bel Hadj Salah, "Real-time deformation of structure using finite element and neural networks in virtual reality applications," *Finite Elements in Analysis and Design*, vol. 42, pp. 985–991, July 2006.

[43] M. Connell, "A framework for immersive fem visualisation using transparent object communication in a distributed network environment," *Advances in Engineering Software*, May 1996.

[44] F. Naets, F. Cosco, and W. Desmet, "Improved human-computer interaction for mechanical systems design through augmented strain/stress visualisation," *International Journal of Intelligent Engineering Informatics (IJIEI)*, vol. 5, no. 1, 2017.

[45] A. Bartezzaghi, M. Cremonesi, N. Parolini, and U. Perego, "An explicit dynamics GPU structural solver for thin shell finite elements," *Computers & Structures*, vol. 154, pp. 29–40, July 2015.

[46] A. Buchau and W. Rucker, "Analysis of a three-phase transformer using comsol multiphysics and a virtual reality environment," *2011 COMSOL Conference*, 2011.

[47] D. Lian, I. Oraifige, and F. R. Hall, "Real-time finite element analysis with virtual hands: An introduction," in *World Society for Computer Graphics*, 2004.

[48] a. P. T. Mehdi Setareh, Doug A. Bowman, "Development of a collaborative design tool for structural analysis in an immersive virtual environment," *Seventh International IBPSA Conference*, 2001.

[49] J. Huang, S. Ong, and A. Nee, "Visualization and interaction of finite element analysis in augmented reality," *Computer-Aided Design*, vol. 84, pp. 1–14, Mar. 2017.

[50] T. Ingrassia and F. Cappello, "VirDe: a new virtual reality design approach," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 3, pp. 1–11, Feb. 2009.

[51] D. Weidlich, S. Scherer, and M. Wabner, "Analyses using vr/ar visualization," *IEEE Computer Graphics and Applications*, vol. 28, no. 5, pp. 84–86, 2008.

[52] S. Yao, "Teaching structural analysis with a mobile augmented reality application," *Master's thesis, Iowa State University, Ames, USA*, 2018.

[53] C. Quesada, D. Gonzalez, and I. Alfaro, "Real-time simulation techniques for augmented learning in science and engineering," *The Visual Computer*, vol. 32, pp. 1465–1479, 2016.

[54] E. Pruna, M. Rosero, R. Pogo, I. Escobar, and J. Acosta, "Virtual Reality as a Tool for the Cascade Control Learning," in *Augmented Reality, Virtual Reality, and Computer Graphics* (L. T. De Paolis and P. Bourdot, eds.), vol. 10850, pp. 243–251, Cham: Springer International Publishing, 2018.

[55] R. L. Hite, M. G. Jones, G. M. Childers, M. Ennes, K. Chesnutt, M. Pereyra, and E. Cayton, "Investigating Potential Relationships Between Adolescents' Cognitive Development and Perceptions of Presence in 3-D, Haptic-Enabled, Virtual Reality Science Instruction," *Journal of Science Education and Technology*, vol. 28, pp. 265–284, June 2019.

[56] K. D'Souza, "Technology to Transform Lives:," *NAEFMS BENCHMARK*, p. 6, 2015.

[57] R. D. Cook, D. S. Malkus, and M. Plesha, "Concepts and applications of finite element analysis," *New York: Wiley 3rd ed.*, 1989.

[58] K. K. Choi and N.-H. Kim, "Structural sensitivity analysis and optimization 1: linear systems.," *Springer Science Business Media*, 2006.

[59] C. A. Ribas, "Finite element analysis of stresses in beam structures," *Master's thesis, AALTO University, Espoo, Finland*, 2011.

[60] J. Whitcomb, "Introduction to classical virtual work and finite elements," *Aero 306 Structural Analysis II*, 2018.

[61] B. R. Bielefeldt, "Multiobjective topology optimization for preliminary design using graph theory and l-system languages," *Doctoral Dissertation, Texas A&M University, College Station, United States*, 2020.

[62] D. L. Logan, "A first course in the finite element mathod," Thomson, 2007.

[63] D. Kincaid, D. Kincaid, and E. Cheney, "Numerical analysis: Mathematics of scientific computing," *American Mathematical Soc.*, 2009.

[64] A. Kaw, *Numerical Methods with Applications: Abridged*. 2009.

[65] ABAQUS, *Beam element library*, 2016.

[66] PolymerDatabase, *Typical Poisson's ratios of polymers at room temperature*, 2015.

[67] D. P. Devine, "Students teach students with the west point bridge designer," *Journal of Professional Issues in Engineering Education and Practice*, 2006.

[68] S. P. Timoshenko and J. M. Gere, "Theory of elastic sability," *Courier Corporation*, 2009.

[69] T. Djukic, V. Mandic, and N. Filipovic, "Virtual reality aided visualization of fluid flow simulations with application in medical education and diagnostics," *Computers in Biology and Medicine*, vol. 43, pp. 2046–2052, Dec. 2013.

APPENDIX A

FEA SOLVER IMPLEMENTATION IN $C\#$

Starts on next page.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using MathNet.Numerics.LinearAlgebra;
using VRTK;
using UnityEngine.Assertions;

public class Solver : MonoBehaviour
{
    List<Dictionary<int, object>> csvData; //The List containing
    the dictionary of Nodes/Elems with their attributes.
    [Tooltip("Use the exact name of the file without the .csv file
    designator")]
    public string csvFilename; // This should be in placed in the
    Assets/Resources folder
    public float ScaleFactor = 1.0f; // A constant that only
    changes with a change in scale by the user
    public List<NodeClass> nodeList = new List<NodeClass>(); //The
    List containing all defined NodeClass nodes
    public List<ElemClass> elemList = new List<ElemClass>(); //The
    List containing all defined ElemClass elements
    public List<Vector3> defElemList; //The List containing the
    locations of all the deformed elements
    NodePropsOld NodePropsScript; //A reference to the NodeProps
    script that helps initialize all the nodes correctly
    public GameObject newNode; //A reference to the undeformed node
    GameObject prefab
    public GameObject newElem; //A reference to the undeformed elem
    GameObject prefab
    [Tooltip("The GameObject parent containing all the Nodes.")]
    public GameObject nodeParent;
    [Tooltip("The GameObject parent containing all the Elements.")]
    public GameObject elemParent;
    public GameObject defNode; //A reference to the deformed node
    GameObject prefab
    [Tooltip("The GameObject parent containing all the deformed
    Nodes.")]
    public GameObject defNodeParent;
    public GameObject defElem; //A reference to the deformed elem
    GameObject prefab
    [Tooltip("The GameObject parent containing all the deformed
    Elems.")]
    public GameObject defElemParent;
    int numNodes;  //A local int variable that keeps track of the
    number of nodes in the scene
    int numElem; //A local int variable that keeps track of the
    number of elements in the scene
    public double YoungsModulus = 1.5E9; // A constant that only
    changes with a change in material by the user
    public double Radius = 0.01; // A constant that only changes
    with a change in radius by the user
    public double MaxStress = 1E6; // A constant that only changes
    with a change in criteria by the user
    public ElemMaker ElemMakerScript; //A reference to the
    ElemMaker script that helps create elems
    [Tooltip("The GameObject containing the error that gets
    displayed when computation is impossible.")]
    public GameObject errorCanvas;
    System.Diagnostics.Stopwatch StopWatch = new
    System.Diagnostics.Stopwatch(); //A local reference to the
    Stopwatch script
    string readType; //A local variable that keeps track of whether
    nodes or elements are being read by the CSVReader
    public NodeCanvas NodeCanvasScript; //A reference to the
    NodeCanvas script that keeps track of nodal data


```

```
44
45
46      public class ElemClass : IComparable<ElemClass>
47      {
48          // This is a class that enforces a standard across elements
        •   and keeps their attributes organized.
49          public GameObject nodeA; // References the beginning node
        •   of the element
50          public GameObject nodeB; // References the end node of the
        •   element
51          public GameObject GO; // An Unity standard; GO refers to
        •   the element's GameObject that is visualized
52
53          public int CompareTo(ElemClass compareElem)
54          {
55              // This method allows the use of the Sort() function
        •       which helps organize the elemList to make sure there
        •       are no gaps in numbering.
56              int elemNum = Convert.ToInt32(this.GO.name.Remove(0,
        •       4)); //Parses the GO name to remove "Elem" from the
        •       name and returns just the element number
57              int compareNum =
        •       Convert.ToInt32(compareElem.GO.name.Remove(0, 4)); //
        •       Parses the GO name of the compared elem to remove
        •       "Elem" and returns just the element number
58              if (compareElem == null) // A null value means that
        •       this object is greater and there is a numbering jump
        •       that needs to be rectified at the source
59                  return 1;
60              else // No numbering gap, continue ordering the list
61                  return elemNum.CompareTo(compareNum);
62          }
63      }
64
65      public class NodeClass : IComparable<NodeClass>
66      {
67          // This is a class that enforces a standard across nodes
        •   and keeps their attributes organized.
68          public GameObject GO; // An Unity standard; GO refers to
        •   the node's GameObject that is visualized
69          public bool fixedX; // Is translation possible in the x-
        •   direction?
70          public bool fixedY; // Is translation possible in the y-
        •   direction?
71          public bool fixedZ; // Is translation possible in the z-
        •   direction?
72          public bool xRot; // Is rotation possible in the x-
        •   direction?
73          public bool yRot; // Is rotation possible in the y-
        •   direction?
74          public bool zRot; // Is rotation possible in the z-
        •   direction?
75          public double forceX; // How much force in Newtons is
        •   acting in the x-direction?
76          public double forceY; // How much force in Newtons is
        •   acting in the y-direction?
77          public double forceZ; // How much force in Newtons is
        •   acting in the z-direction?
78
79          public int CompareTo(NodeClass compareNode)
80          {
81              // This method allows the use of the Sort() function
        •       which helps organize the nodeList to make sure there
        •       are no gaps in numbering.
82              int nodeNum = Convert.ToInt32(this.GO.name.Remove(0,
        •       4)); //Parses the GO name to remove "Node" from the
        •       name and returns just the node number
83              int compareNum =
        •       Convert.ToInt32(compareNode.GO.name.Remove(0, 4)); //
```

```
            Convert.ToInt32(compareNode.GO.name.Remove(0, 4)); //
            Parses the GO name of the compared elem to remove
            "Node" and returns just the node number
            if (compareNode == null) // A null value means that
            this object is greater and there is a numbering jump
            that needs to be rectified at the source
                return 1;
            else // No numbering gap, continue ordering the list
                return nodeNum.CompareTo(compareNum);
        }
    }

    public void PrintElemNames()
    {
        //A Debugging test that prints the names of each elem in
        elemList
        foreach (ElemClass elem in elemList)
        {
            Debug.Log(elem.GO.name);
        }
    }


    void RemoveElems()
    {
        //A Debugging test that checks if any elems in elemList are
        null (which means they have been deleted elsewhere) and
        then proceeds to delete the element from elemList
        foreach (ElemClass elem in elemList)
        {
            GameObject tempElem = GameObject.Find(elem.GO.name);
            if (tempElem == null)
            {
                Debug.Log(elem.GO.name + " has been removed");
                elemList.Remove(elem);
            }
        }

    }

    public void ControlElemMaker(bool active)
    {
        // This function controls whether or not the ElemMaker
        script is active or inactive, based on user selection.
        ElemMakerScript.enabled = active;
        //Debug.Log(ElemMakerOn.enabled);
    }

    void Awake()
    {
        // This is the first function to run when the simulation is
        initialized. It parses the csv file and creates the nodes/
        elems defined there.
        if (Resources.Load(csvFilename) != null)
        {
            csvData = CSVReader.Read(csvFilename);  // the filename
            in which the structural information is located, must be
            a .csv
        }
        else
        {
            print("CRITICAL ERROR: CSV FILE NOT FOUND! Analysis can
            not be performed"); // If no csv file is defined, the
            application cannot run, so it force quits
            Application.Quit();
            this.enabled = false;
        }

        for (int i = 0; i < csvData.Count; i++)
```

```
138         {
139             // This loop iterates through every line in the csv file
140             if (Convert.ToString(csvData[i][0]) == "*Node")
141             {
142                 // if *Node is found, Changes readType to Node
143                 readType = Convert.ToString(csvData[i][0]);
144                 Debug.Log("Starting Node Analysis");
145 
146             }
147             else if (Convert.ToString(csvData[i][0]) == "*Element")
148             {
149                 // if *Element is foun, changes readType to Element
150                 readType = Convert.ToString(csvData[i][0]);
151                 Debug.Log("Starting Element Analysis");
152             }
153             else
154             {
155                 if (readType == "*Node")
156                 {
157                     // Reads node properties from .csv file and
  •                  iteratively adds NodeClass nodes to the nodeList
158                     int nodeNum = (Convert.ToInt32(csvData[i][0]));
159                     string nodeName = "Node" + nodeNum.ToString();
160                     float x = (Convert.ToSingle(csvData[i][1]));
161                     float y = (Convert.ToSingle(csvData[i][2]));
162                     float z = (Convert.ToSingle(csvData[i][3]));
163 
164                     Vector3 newPos = new Vector3(x, y, z);
165 
166                     GameObject Node = Instantiate(newNode, newPos,
  •                  Quaternion.identity, nodeParent.transform);
167                     Node.name = nodeName;
168 
169                     NodePropsOld NodePropsScript =
  •                  Node.GetComponent<NodePropsOld>();
170                     int xDOF = Convert.ToInt32(csvData[i][4]);
171                     int yDOF = Convert.ToInt32(csvData[i][5]);
172                     int zDOF = Convert.ToInt32(csvData[i][6]);
173                     int xRot = Convert.ToInt32(csvData[i][7]);
174                     int yRot = Convert.ToInt32(csvData[i][8]);
175                     int zRot = Convert.ToInt32(csvData[i][9]);
176                     double fx = Convert.ToDouble(csvData[i][10]);
177                     double fy = Convert.ToDouble(csvData[i][11]);
178                     double fz = Convert.ToDouble(csvData[i][12]);
179 
180                     if (xDOF == 1)
181                     {
182                         NodePropsScript.fixedX = true;
183                     }
184 
185                     if (yDOF == 1)
186                     {
187                         NodePropsScript.fixedY = true;
188                     }
189 
190                     if (zDOF == 1)
191                     {
192                         NodePropsScript.fixedZ = true;
193                     }
194 
195                     if (xRot == 1)
196                     {
197                         NodePropsScript.xRot = true;
198                     }
199 
200                     if (yRot == 1)
201                     {
202                         NodePropsScript.yRot = true;
```

```
                    }

                    if (zRot == 1)
                    {
                        NodePropsScript.zRot = true;
                    }

                    NodePropsScript.forceX = fx;
                    NodePropsScript.forceY = fy;
                    NodePropsScript.forceZ = fz;

                    var tempNode = new NodeClass()
                    {
                        GO = Node,
                        fixedX = NodePropsScript.fixedX,
                        fixedY = NodePropsScript.fixedY,
                        fixedZ = NodePropsScript.fixedZ,
                        xRot = NodePropsScript.xRot,
                        yRot = NodePropsScript.yRot,
                        zRot = NodePropsScript.zRot,
                        forceX = NodePropsScript.forceX,
                        forceY = NodePropsScript.forceY,
                        forceZ = NodePropsScript.forceZ
                    };

                    nodeList.Add(tempNode);
                    Debug.Log(nodeName + " was added to nodeList");
                    NodePropsScript.DisplayForce();
                    NodePropsScript.DisplaySupports();

                }
                else if (readType == "*Element")
                {
                    // Reads elem properties from .csv file and
                    iteratively adds EodeClass elems to the elemList
                    int elemNum = Convert.ToInt32(csvData[i][0]);
                    int n1 = Convert.ToInt32(csvData[i][1]);
                    int n2 = Convert.ToInt32(csvData[i][2]);
                    Vector3 posA =
                    nodeList[n1].GO.transform.position;
                    Vector3 posB =
                    nodeList[n2].GO.transform.position;
                    Vector3 middlePos = Vector3.Lerp(posA, posB,
                    0.5f); // middle position between the origin
                    node and the moveable node
                    var dir = posB - posA;
                    var myRot =
                    Quaternion.FromToRotation(Vector3.up, dir);
                    GameObject element = Instantiate(newElem,
                    middlePos, myRot, elemParent.transform);
                    element.name = "Elem" + elemNum;
                    element.transform.localScale = new
                    Vector3((float)Radius * 2f,
                    (Vector3.Distance(posA, posB)) * 0.5f,
                    (float)Radius * 2f);

                    var tempElem = new ElemClass
                    {
                        nodeA = nodeList[n1].GO,
                        nodeB = nodeList[n2].GO,
                        GO = element
                    };
                    elemList.Add(tempElem);
                }
            }
        }

    }
```

```
261
262     void Start()
263     {
264         // Start is called before the simulation starts rendering
  •         to the headset
265         UpdateNodeCanvas();
266     }
267
268     public void UpdateNodeCanvas()
269     {
270         // This function updates the NodeCanvas bounds to be in the
  •         user-defined criteria before run time
271         NodeCanvasScript.maxForce = (float)(0.001 * YoungsModulus *
  •         Math.PI * Math.Pow(Radius, 2));
272         NodeCanvasScript.UpdateMaxForce();
273     }
274
275     public void UpdateNodeProps()
276     {
277         // This function updates all node properties of every node
  •         in nodeList if there has been a change.
278         numNodes = nodeParent.transform.childCount; // Updates the
  •         local variable that keeps track of the total number of
  •         nodes in the simulation
279         foreach (NodeClass node in nodeList)
280         {
281             NodePropsScript = node.GO.GetComponent<NodePropsOld>();
282             node.fixedX = NodePropsScript.fixedX;
283             node.fixedY = NodePropsScript.fixedY;
284             node.fixedZ = NodePropsScript.fixedZ;
285             node.forceX = NodePropsScript.forceX;
286             node.forceY = NodePropsScript.forceY;
287             node.forceZ = NodePropsScript.forceZ;
288         }
289     }
290
291     public void ResetDeformed()
292     {
293         // Iteratively deleting GameObjects in Unity is difficult.
294         // This function is an implementation which adds all the
  •         deformed GameObjects (nodes and elems) into a list and
  •         destroys the whole list.
295         int nodeI = 0;
296         int elemI = 0;
297         GameObject[] allDefNodes = new
  •         GameObject[defNodeParent.transform.childCount];
298         GameObject[] allDefElems = new
  •         GameObject[defElemParent.transform.childCount];
299
300         foreach (Transform child in defNodeParent.transform)
301         {
302             allDefNodes[nodeI] = child.gameObject;
303             nodeI += 1;
304         }
305
306         foreach (Transform child in defElemParent.transform)
307         {
308             allDefElems[elemI] = child.gameObject;
309             elemI += 1;
310         }
311
312         foreach (GameObject child in allDefNodes)
313         {
314             DestroyImmediate(child.gameObject);
315         }
316
317         foreach (GameObject child in allDefElems)
318         {
319             DestroyImmediate(child.gameObject);
```

```
320         }
321
322     }
323
324     public void UpdateStruct()
325     {
326         // This function updates each elem in the elemList to its
    •    new position if there were any changes to the structure.
327         foreach (ElemClass elem in elemList)
328         {
329             GameObject elemUsed = GameObject.Find(elem.GO.name);
330             int el1 =
    •        Convert.ToInt32(elem.nodeA.gameObject.name.Remove(0,
    •        4)); // The number associated with nodeA
331             int el2 =
    •        Convert.ToInt32(elem.nodeB.gameObject.name.Remove(0,
    •        4)); // The number associated with nodeB
332             if (elemUsed != null) // If element exist, update its
    •        location
333             {
334                 Vector3 middlePos =
    •            Vector3.Lerp(elem.nodeA.transform.position,
    •            elem.nodeB.transform.position, 0.5f); // middle
    •            position between the start node and the end node
335                 elemUsed.transform.position = middlePos;
336                 var dir = elem.nodeB.transform.position -
    •            elem.nodeA.transform.position;
337                 elemUsed.transform.rotation =
    •            Quaternion.FromToRotation(Vector3.up, dir);
338                 elemUsed.transform.localScale = new
    •            Vector3((float)Radius * 2f,
    •            (Vector3.Distance(elem.nodeA.transform.position,
    •            elem.nodeB.transform.position)) * 0.5f,
    •            (float)Radius * 2f);
339             }
340             else // If element doesn't exist, make a new element
341             {
342                 MakeElem(elem.nodeA.transform.position,
    •            elem.nodeA.transform.position, elemParent,
    •            elem.GO.name, 0f);
343             }
344         }
345
346         foreach (NodeClass node in nodeList)
347         {
348             // Updates the Boundary Conditions and Forces for all
    •        nodes
349             node.GO.GetComponent<NodeProps>().DisplaySupports();
350             node.GO.GetComponent<NodeProps>().DisplayForce();
351         }
352     }
353
354     public void UpdateRadius()
355     {
356         // This function gets called when the user changes the
    •        radius using the radius slider
357         // It iteratively goes through every elem in elemList and
    •        updates its size
358         foreach (ElemClass elem in elemList)
359         {
360             GameObject elemUsed = GameObject.Find(elem.GO.name);
361             int el1 =
    •        Convert.ToInt32(elem.nodeA.gameObject.name.Remove(0,
    •        4));
362             int el2 =
    •        Convert.ToInt32(elem.nodeB.gameObject.name.Remove(0,
    •        4));
363             if (elemUsed != null)
```

```
            {
                elemUsed.transform.localScale = new
                Vector3((float)Radius * 2f,
                (Vector3.Distance(elem.nodeA.transform.position,
                elem.nodeB.transform.position)) * 0.5f,
                (float)Radius * 2f);
            }
        }
    }

    void Update()
    {
        // Update is called once per frame but is not utilized as a
        function in this simulation

    }

    void MakeElem(Vector3 posA, Vector3 posB, GameObject parent,
    String name, float stress)
    {
        // This function instantiates a deformed element with a
        color that corresponds to the amount of stress in the
        elements
        // The color spectrum goes from blue (unstressed) to purple
        (partially stressed) to red (highly stressed) and then
        white (overstressed)
        Vector3 middlePos = Vector3.Lerp(posA, posB, 0.5f); //
        middle position between the origin node and the moveable
        node
        var dir = posB - posA;
        var myRot = Quaternion.FromToRotation(Vector3.up, dir);
        GameObject element = Instantiate(defElem, middlePos, myRot,
        parent.transform);
        Renderer elemRend = element.GetComponent<Renderer>();

        if (stress > 1)
        {
            stress = 1f;
            elemRend.material.color = new Color(1f, 1f, 1f); //
            white
        }
        else if (stress < 0)
        {
            stress = 0.0f;
            elemRend.material.color = new Color(0, 0, 1); // blue
        }
        else
        {
            elemRend.material.color = new Color(stress, 0, 1 -
            stress); // color gradient between red and blue
        }

        element.name = name;
        element.transform.localScale = new Vector3((float)Radius *
        2f, (Vector3.Distance(posA, posB)) * 0.5f, (float)Radius *
        2f);
    }

    public void RenameElems()
    {
        // This function sorts the elemList by number and renames
        them to be in the correct order with no gaps in numbering
        elemList.Sort();
        int elemCount = elemParent.transform.childCount;
        for (int i = 0; i < elemCount; i++)
        {
            elemList[i].GO.name = "Elem" + i.ToString();
        }
```

```
414     }
415
416     public void RenameNodes()
417     {
418         // This function sorts the nodeList by number and renames
  •      them to be in the correct order with no gaps in numbering
419         nodeList.Sort();
420         int nodeCount = nodeList.Count;
421         for (int i = 0; i < nodeCount; i++)
422         {
423             nodeList[i].GO.name = "Node" + i.ToString();
424         }
425     }
426
427     void TestNodeListLengths()
428     {
429         // This function compares the deformed nodeList and the
  •      undeformed nodeLists and make sure they have the same
  •      number of nodes
430         int undefNodeCount = nodeParent.transform.childCount;
431         int defNodeCount = defNodeParent.transform.childCount;
432
433         Assert.AreEqual(undefNodeCount, defNodeCount);
434     }
435
436     void TestElemListLengths()
437     {
438         // This function compares the deformed elemList and the
  •      undeformed elemLists and make sure they have the same
  •      number of elements
439         int undefElemCount = elemParent.transform.childCount;
440         int defElemCount = defElemParent.transform.childCount;
441
442         Assert.AreEqual(undefElemCount, defElemCount);
443     }
444
445     public void DoAnalysis()
446     {
447         // This is the quintessential function within this script
  •      that does all of the finite element analysis
448         ResetDeformed(); // Deletes any old cached data before
  •      conducting analysis
449         RenameElems(); // Orders the elemList to remove analysis
  •      bugs caused by numbering gaps
450         UpdateNodeProps(); // Updates the nodal properties to make
  •      sure there have been no unregistered changes
451         StopWatch.Start(); // Starts the timing of the analysis
  •      section
452         var M = Matrix<double>.Build; // Simplifies the future
  •      scripting through this M variable which represents matrix
  •      creation
453         var V = Vector<double>.Build; // Simplifies the future
  •      scripting through this V variable which represents vector
  •      creation
454         var Lambda = M.Dense(3, 3, 0d); // Lambda is a 3 by 3
  •      matrix initialized with doubles that have a value of 0.
455         numNodes = nodeParent.transform.childCount; // Updates
  •      local node count
456         numElem = elemParent.transform.childCount; // Updates local
  •      element count
457         int gDOF = numNodes * 6; // Global Degreees of Freedom
  •      (DOF) = 6 DOF per node times number of Nodes
458         var kGlobal = M.Dense(gDOF, gDOF, 0d); // represents global
  •      stiffness of the structure and is a gDOF by gDOF matrix
  •      initialized with doubles that have a value of 0.
459         var fGlobal = M.Dense(gDOF, 1, 0d); // represents global
  •      forces of the structure and is a gDOF by 1 matrix
  •      initialized with doubles that have a value of 0.
460         var elemDOFs = M.Dense(numElem, 12, 0d); // represents
```

```
        var ElemDofs = M.Dense(numElem, 12, 0d); // represents
        total DOFs of the structure and is a numElem by 12 (DOF per
        elem) matrix initialized with doubles that have a value of
        0.
        bool structError = false; // A boolean that keeps track of
        whether or not the structure can be analyzed

        var ShearModulus = YoungsModulus / (2 * (1 + 0.43)); // A
        constant derived from the Youngs Modulus

        //Circular Cross Section Case
        var A = Math.PI * Math.Pow(Radius, 2);
        var Iz = Math.PI * Math.Pow(Radius, 4) * 0.25;
        var Iy = Math.PI * Math.Pow(Radius, 4) * 0.25;
        var J = Iz + Iy;
        var EA = YoungsModulus * A;

        //Square Cross Section Case
        /*var t = 2e-3;
        var w = 0.01;
        var A = t*w;
        var Iz = w * Math.Pow(t, 3) /12;
        var Iy = t * Math.Pow(w, 3) /12;
        var J = Iy + Iz;
        var EA = E * A;*/

        int i; // initialized here for utilization in the foreach
        loop below
        foreach (NodeClass node in nodeList)
        {
            // Applies the X,Y,Z forces from each node to the
            global force vector
            i = Convert.ToInt32(node.GO.name.Remove(0, 4));
            fGlobal[(i * 6), 0] = node.forceX;
            fGlobal[(i * 6) + 1, 0] = node.forceY;
            fGlobal[(i * 6) + 2, 0] = node.forceZ;
            // fGlobal[(i * 6) + 2, 0] is Moment force in x-
            direction, etc but Moment forces are not implemented
            for my thesis but could be implemented in the future
        }

        foreach (ElemClass elem in elemList)
        {
            // This foreach loop computes the element stiffness for
            each elem in elemList and appropriately adds it to the
            global stiffness of the structure
            double n1 = Convert.ToDouble(elem.nodeA.name.Remove(0,
            4)); // start node of element
            double n2 = Convert.ToDouble(elem.nodeB.name.Remove(0,
            4)); // end node of element

            Vector<double> elemDOF = V.DenseOfArray(new double[]
            {(n1+1)*6 - 6, (n1 + 1) * 6 - 5, (n1 + 1) * 6 - 4, (n1
            + 1) * 6 - 3, (n1 + 1) * 6 - 2, (n1 + 1) * 6 - 1,
            (n2+1)*6 - 6, (n2 + 1) * 6 - 5, (n2 + 1) * 6 - 4, (n2 +
            1) * 6 - 3, (n2 + 1) * 6 - 2, (n2 + 1) * 6 - 1}); //
            DOF vector of element

            double elemLen =
            Math.Sqrt(Math.Pow(elem.nodeA.transform.position.x -
            elem.nodeB.transform.position.x, 2) +
            Math.Pow(elem.nodeA.transform.position.y -
            elem.nodeB.transform.position.y, 2)
                + Math.Pow(elem.nodeA.transform.position.z -
                elem.nodeB.transform.position.z, 2)); // length of
                element


            double mass = elemLen * A * 1175; // mass of element
```

```
            double k1 = EA / elemLen;
            double k2 = 12 * YoungsModulus * Iz / Math.Pow(elemLen,
            3);
            double k3 = 6 * YoungsModulus * Iz / Math.Pow(elemLen,
            2);
            double k4 = 4 * YoungsModulus * Iz / elemLen;
            double k5 = 2 * YoungsModulus * Iz / elemLen;
            double k6 = 12 * YoungsModulus * Iy / Math.Pow(elemLen,
            3);
            double k7 = 6 * YoungsModulus * Iy / Math.Pow(elemLen,
            2);
            double k8 = 4 * YoungsModulus * Iy / elemLen;
            double k9 = 2 * YoungsModulus * Iy / elemLen;
            double k10 = ShearModulus * J / elemLen;

            var kElem = M.DenseOfArray(new double[,]
            {
                {k1,0,0,0,0,0,-k1,0,0,0,0,0},
                {0,k2,0,0,0,k3,0,-k2,0,0,0,k3},
                {0,0,k6,0,-k7,0,0,0,-k6,0,-k7,0},
                {0,0,0,k10,0,0,0,0,0,-k10,0,0},
                {0,0,-k7,0,k8,0,0,0,k7,0,k9,0},
                {0,k3,0,0,0,k4,0,-k3,0,0,0,k5},
                {-k1,0,0,0,0,0,k1,0,0,0,0,0},
                {0,-k2,0,0,0,-k3,0,k2,0,0,0,-k3},
                {0,0,-k6,0,k7,0,0,0,k6,0,k7,0},
                {0,0,0,-k10,0,0,0,0,0,k10,0,0},
                {0,0,-k7,0,k9,0,0,0,k7,0,k8,0},
                {0,k3,0,0,0,k5,0,-k3,0,0,0,k4}
            }); // stiffness matrix of local element

            //Debug.Log(kElem.ToString());
            // Initialization of the Lambda matrix for use within
            the Transformation matrix T
            if (elem.nodeA.transform.position.x ==
            elem.nodeB.transform.position.x &&
            elem.nodeA.transform.position.y ==
            elem.nodeB.transform.position.y)
            {
                if (elem.nodeB.transform.position.z >
                elem.nodeA.transform.position.z)
                {
                    Lambda = M.DenseOfArray(new double[,] { { 0, 0,
                    1 }, { 0, 1, 0 }, { -1, 0, 0 } });
                }
                else
                {
                    Lambda = M.DenseOfArray(new double[,] { { 0, 0,
                    -1 }, { 0, 1, 0 }, { 1, 0, 0 } });
                }
            }
            else
            {
                double CXx = (elem.nodeB.transform.position.x -
                elem.nodeA.transform.position.x) / elemLen;
                double CYx = (elem.nodeB.transform.position.y -
                elem.nodeA.transform.position.y) / elemLen;
                double CZx = (elem.nodeB.transform.position.z -
                elem.nodeA.transform.position.z) / elemLen;
                double D = Math.Sqrt(Math.Pow(CXx, 2) +
                Math.Pow(CYx, 2));
                double CXy = -CYx / D;
                double CYy = -CXx / D;
                double CZy = 0d;
                double CXz = -CXx * CZx / D;
                double CYz = -CYx * CZx / D;
                double CZz = D;
                Lambda = M.DenseOfArray(new double[,] { { CXx, CYx,
```

```
                CZx }, { CXy, CYy, CZy }, { CXz, CYz, CZz } });
            }

            var zeros39 = M.Dense(3, 9, 0d); // 3 by 9 Matrix of
            zeros
            var zeros36 = M.Dense(3, 6, 0d); // 3 by 6 Matrix of
            zeros
            var zeros33 = M.Dense(3, 3, 0d); // 3 by 3 Matrix of
            zeros
            var row1 = Lambda.Append(zeros39);
            var row2 = zeros33.Append(Lambda.Append(zeros36));
            var row3 = zeros36.Append(Lambda.Append(zeros33));
            var row4 = zeros39.Append(Lambda);
            var T = row1.Stack(row2.Stack(row3.Stack(row4))); //
            Transformation matrix T composed of staggered Lambda
            matrices

            var K0 = T.Transpose() * kElem; // intermediary
            variable for kTransformed
            var kTransformed = K0 * T; // broke kTransformed
            equation into 2 lines due to Unity errors otherwise

            for (int a = 0; a < 12; a++)
            {
                for (int b = 0; b < 12; b++)
                {
                    kGlobal[(int)(elemDOF[a]), (int)(elemDOF[b])] =
                    kGlobal[(int)(elemDOF[a]), (int)(elemDOF[b])] +
                    kTransformed[a, b]; // Add local elem stiffness
                    to global stiffness
                }
            }
            //Iterate for every elem in elemList
        }


        foreach (NodeClass node in nodeList)
        {
            // This foreach loop iteratively checks if any boundary
            conditions are fixed for any nodes, and zeroes out the
            related rows/columns.
            int nodeNum = Convert.ToInt32(node.GO.name.Remove(0,
            4));

            if (node.fixedX == true)
            {
                kGlobal.SetRow((6 * nodeNum), V.Dense(gDOF));
                kGlobal.SetColumn((6 * nodeNum), V.Dense(gDOF));
                kGlobal[(6 * nodeNum), (6 * nodeNum)] = 1;
                fGlobal[(6 * nodeNum), 0] = 0;
            }

            if (node.fixedY == true)
            {
                kGlobal.SetRow((6 * nodeNum) + 1, V.Dense(gDOF));
                kGlobal.SetColumn((6 * nodeNum) + 1, V.Dense(gDOF));
                kGlobal[(6 * nodeNum) + 1, (6 * nodeNum) + 1] = 1;
                fGlobal[(6 * nodeNum) + 1, 0] = 0;
            }

            if (node.fixedZ == true)
            {
                kGlobal.SetRow((6 * nodeNum) + 2, V.Dense(gDOF));
                kGlobal.SetColumn((6 * nodeNum) + 2, V.Dense(gDOF));
                kGlobal[(6 * nodeNum) + 2, (6 * nodeNum) + 2] = 1;
                fGlobal[(6 * nodeNum) + 2, 0] = 0;
            }

            if (node.xRot == true)
```

```
614             if (node.xRot == true)
615             {
616                 kGlobal.SetRow((6 * nodeNum) + 3, V.Dense(gDOF));
617                 kGlobal.SetColumn((6 * nodeNum) + 3, V.Dense(gDOF));
618                 kGlobal[(6 * nodeNum) + 3, (6 * nodeNum) + 3] = 1;
619                 fGlobal[(6 * nodeNum) + 3, 0] = 0;
620             }
621 
622             if (node.yRot == true)
623             {
624                 kGlobal.SetRow((6 * nodeNum) + 4, V.Dense(gDOF));
625                 kGlobal.SetColumn((6 * nodeNum) + 4, V.Dense(gDOF));
626                 kGlobal[(6 * nodeNum) + 4, (6 * nodeNum) + 4] = 1;
627                 fGlobal[(6 * nodeNum) + 4, 0] = 0;
628             }
629 
630             if (node.zRot == true)
631             {
632                 kGlobal.SetRow((6 * nodeNum) + 5, V.Dense(gDOF));
633                 kGlobal.SetColumn((6 * nodeNum) + 5, V.Dense(gDOF));
634                 kGlobal[(6 * nodeNum) + 5, (6 * nodeNum) + 5] = 1;
635                 fGlobal[(6 * nodeNum) + 5, 0] = 0;
636             }
637         }
638         //Debug.Log(kGlobal.ToString());
639         //Debug.Log(fGlobal.ToString());
640 
641         //var dGlobal = kGlobal.Inverse() * fGlobal;
642         var dGlobal = kGlobal.Cholesky().Solve(fGlobal); // Solve
  •      the system of linear equations to find displacement,
  •      Cholesky Decomp used instead of Matrix Inversion
643         StopWatch.Stop(); // Stops the analysis timer and outputs
  •      computation time
644         Debug.Log("For a Test of " + numNodes + " nodes, and " +
  •      numElem + " elems:");
645         Debug.Log("Compute Time: " + (StopWatch.Elapsed));
646         StopWatch.Reset();
647         //Debug.Log(dGlobal.ToString());
648 
649 
650         Debug.Log("Node Positions:");
651         int nodeNeeded = 2;
652         Debug.Log("X-pos: " +
  •      nodeList[nodeNeeded].GO.transform.position.x + ", Y-pos: "
  •      + nodeList[nodeNeeded].GO.transform.position.y + ", Z-pos:
  •      " + nodeList[nodeNeeded].GO.transform.position.z);
653         Debug.Log("X-disp: " + dGlobal[(6 * nodeNeeded), 0] + ", Y-
  •      disp: " + dGlobal[(6 * nodeNeeded) + 1, 0] + ", Z-disp: " +
  •      dGlobal[(6 * nodeNeeded) + 2, 0]);
654         Debug.Log(Math.Sqrt(Math.Pow(dGlobal[(6 * nodeNeeded), 0],
  •      2) + Math.Pow(dGlobal[(6 * nodeNeeded) + 1, 0], 2) +
  •      Math.Pow(dGlobal[(6 * nodeNeeded) + 2, 0], 2)));
655 
656         // Test Case implemented below for if there are any
  •      unattached nodes
657         for (int a = 0; a < gDOF; a++)
658         {
659             //Debug.Log(qGlobal[a, 0]);
660             if (double.IsNaN(dGlobal[a, 0]) == true)
661             {
662                 Debug.Log("Error Found, unattached nodes, can't
  •                  solve qGlobal");
663                 structError = true;
664                 break;
665             }
666         }
667 
668         defElemList.Clear();
669         // Display Error Canvas if structure not solved, otherwise
```

```
        display deformed structure
        if (structError == true)
        {
            errorCanvas.SetActive(true);
        }
        else if (structError == false)
        {
            errorCanvas.SetActive(false);
            foreach (NodeClass node in nodeList)
            {
                // Iteratively go through every node in nodeList
                and create a new deformed node at its updated
                location
                int nodeNum =
                Convert.ToInt32(node.GO.name.Remove(0, 4));
                float newX = (float)(node.GO.transform.position.x +
                dGlobal[(6 * nodeNum), 0]);
                float newY = (float)(node.GO.transform.position.y +
                dGlobal[(6 * nodeNum) + 1, 0]);
                float newZ = (float)(node.GO.transform.position.z +
                dGlobal[(6 * nodeNum) + 2, 0]);

                Vector3 newPos = new Vector3(newX, newY, newZ); //
                New position of node after loading

                defElemList.Add(newPos);
                GameObject deformedNode = Instantiate(defNode,
                newPos, Quaternion.identity,
                defNodeParent.transform);
                deformedNode.name = "Def" + node.GO.name;
            }

            foreach (ElemClass elem in elemList)
            {
                //New Implementation with ToggleDeformed()
                int n1 =
                Convert.ToInt32(elem.nodeA.gameObject.name.Remove(0,
                 4));
                int n2 =
                Convert.ToInt32(elem.nodeB.gameObject.name.Remove(0,
                 4));

                int elemNum =
                Convert.ToInt32(elem.GO.name.Remove(0, 4));

                double defElemLen =
                Math.Sqrt(Math.Pow(elem.nodeA.transform.position.x
                - elem.nodeB.transform.position.x, 2.0d)
                    + Math.Pow(elem.nodeA.transform.position.y -
                    elem.nodeB.transform.position.y, 2.0d)
                    + Math.Pow(elem.nodeA.transform.position.z -
                    elem.nodeB.transform.position.z, 2.0d)); //
                    Length of deformed element

                double sigxxTopNodeA = fGlobal[6 * n1, 0] / A -
                fGlobal[(6 * n1) + 5, 0] * Radius / Iz; // Fx/A -
                Mz*r/Iz
                double sigxxBotNodeA = (fGlobal[6 * n1, 0] / A) +
                fGlobal[(6 * n1) + 5, 0] * Radius / Iz;

                double sigyyLeftNodeA = (fGlobal[(6 * n1) + 1, 0] /
                A) - fGlobal[(6 * n1) + 5, 0] * Radius / Iz; // Fy/
                A - Mz*r/Iz
                double sigyyRightNodeA = (fGlobal[(6 * n1) + 1, 0]
                / A) + fGlobal[(6 * n1) + 5, 0] * Radius / Iz;

                double sigxxTopNodeB = (fGlobal[6 * n2, 0] / A) -
                (fGlobal[(6 * n2) + 5, 0] * Radius / Iz);
```

```
                double sigxxBotNodeB = (fGlobal[6 * n2, 0] / A) +
                fGlobal[(6 * n2) + 5, 0] * Radius / Iz;

                double sigyyLeftNodeB = (fGlobal[(6 * n2) + 1, 0] /
                A) - fGlobal[(6 * n2) + 5, 0] * Radius / Iz; // Fy/
                A - Mz*r/Iz
                double sigyyRightNodeB = (fGlobal[(6 * n2) + 1, 0]
                / A) + fGlobal[(6 * n2) + 5, 0] * Radius / Iz;

                double tstress = Math.Max(Math.Abs(sigxxTopNodeA),
                Math.Abs(sigxxTopNodeB)); // Stress on the top of
                the element [Pa]
                double bstress = Math.Max(Math.Abs(sigxxBotNodeA),
                Math.Abs(sigxxBotNodeB)); // Stress on the bottom
                of the element [Pa]
                double lstress = Math.Max(Math.Abs(sigyyLeftNodeA),
                Math.Abs(sigyyLeftNodeB)); // Stress on the left of
                the element [Pa]
                double rstress =
                Math.Max(Math.Abs(sigyyRightNodeA),
                Math.Abs(sigyyRightNodeB)); // Stress on the right
                of the element [Pa]

                float stressVal = (float)Math.Max(Math.Max(tstress,
                bstress), Math.Max(lstress, rstress)) /
                (float)MaxStress; // A value between 0 and 1 that
                decides what to color the deformed element
                MakeElem(defElemList[n1], defElemList[n2],
                defElemParent, "Def" + elem.GO.name, stressVal);
            }
            TestNodeListLengths();
            TestElemListLengths();
        }
    }
}

```

APPENDIX B

STUDY OF VISUALIZATION SPEEDS IN VIRTUAL REALITY

A study was conducted to understand the visualization of the entire structural deformation process. A structure was created where cubes would be stacked upwards depending on the number of nodes in the structure. This study ranges from 8 nodes (1 cube) to 100 nodes (24 cubes) with exactly the same boundary and loading conditions. The time it takes to visualize the deformed version of this structure was found assuming either a known displacement or a displacement computed by the FEA solver. The results from this study are visualized in Figure B.1. The data shows that the structure with the computed displacement takes significantly more time to visualize than the structure with a known predefined displacement. The difference in time between the two lines represents the time taken by the process of setting up and solving $[\boldsymbol{K}]\{\boldsymbol{D}\} = \{\boldsymbol{F}\}$. It is recommended that future work involve a multi-threaded approach that the computational process can be off-loaded to, and only visualizing the deformed structure when the analysis is completed.

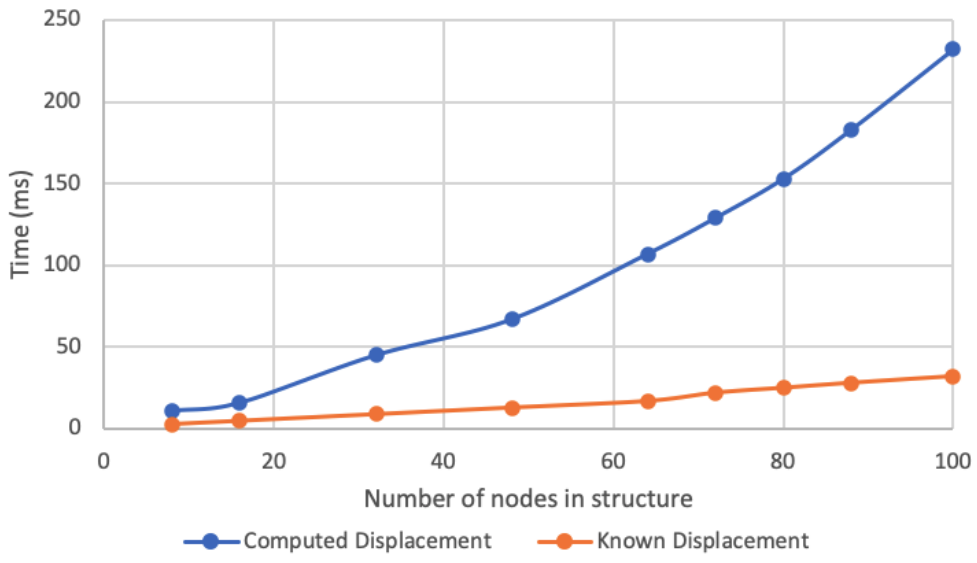


Figure B.1: Graph comparing solver speeds at different complexities with known and computed displacements